SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

THE ROLE OF THE HOST IN A COOPERATING MAINFRAME AND WORKSTATION ENVIRONMENT

FINAL REPORT
VOLUMES I AND II

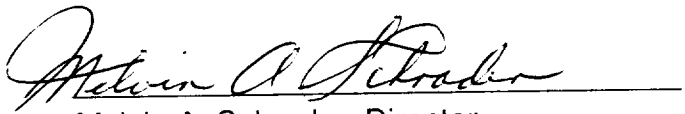
NASA Grant No. NAG 9-341
SwRI Project No. 05-2769

Prepared by:
Antone Kusmanoff, Ph.D.
Nancy L. Martin

Prepared for:
NASA
Johnson Space Center
Houston, TX 77058

October 12, 1989

Approved:



Melvin A. Schrader, Director
Data Systems Science and
Technology Department

THE ROLE OF THE HOST IN A COOPERATING MAINFRAME AND WORKSTATION ENVIRONMENT

Volume I

NASA Grant Number NAG 9-341
SwRI Project No. 05-2769

Submitted to:
NASA-Johnson Space Center
Houston, Texas

Prepared by:
Antone Kusmanoff, Ph.D.
Nancy L. Martin

Southwest Research Institute
6220 Culebra Rd., P.O. Box 28510
San Antonio, TX 78228-0510

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd.)

1.0 INTRODUCTION

1.1 Purpose

In recent years, advancements made in computer systems have prompted a move from centralized computing based on timesharing a large mainframe computer to distributed computing based on a connected set of engineering workstations. A major factor in this advancement is the increased performance and lower cost of engineering workstations. Because of this, workstations have gradually moved from the research lab into the operational rooms as supplements to the user environment. Many complex systems which were previously dependent on the mainframe to do all of the computation and display of information are now using the mainframe as a data acquisition unit and have moved several of the computation and display demands to workstations connected to the mainframe (host) over a Local Area Network (LAN).

The shift to distributed computing from centralized computing has led to challenges associated with the residency of application programs within the system. When there is only one centralized mainframe computer, it handles all data and processing requests for the users. With the advent of engineering workstations, there are now multiple processing units connected together, each accountable for handling their own user requirements. In a combined system of multiple engineering workstations attached to a mainframe host, the question arises as to how does a system designer assign applications between the larger mainframe host and the smaller, yet powerful, workstation. In order to answer this question and develop an effective distributed processing system, a system designer must know the requirements for the types of operations which are to be performed. That is, the designer must know each application's operational requirements for memory, timing, communication, software development features, reliability, the level of user interaction, and coupling.

The purpose of this document is to analyze the concepts related to real-time data processing and display systems which use a host mainframe and a number of engineering workstations interconnected by a LAN. In most cases, distributed systems can be classified as having a single function or multiple functions and as executing programs in real-time or nonreal-time. In a system of multiple computers, the degree of autonomy of the computers is important; a system with one master control computer generally differs in reliability, performance, and complexity from a system in which all computers share the control. This research is concerned with generating general criteria principles for software residency decisions (host or workstation) for a diverse yet coupled group of users (the clustered workstations) which may need the use of a shared resource (the mainframe) to perform their functions.

1.2 Research Focus

The focus of this research is to develop a set of general questions which should be used as guidelines when attempting to determine the residency

of application programs in a distributed computing system. For this research, characteristics of a system containing one mainframe computer connected to numerous engineering workstations will be investigated. Although processing systems included in a distributed system are typically designated as "hosts", the mainframe computer will be the only computer in this document referred to as a host. An additional characterization of the system is that the workstations may be further clustered to perform a particular function. Each workstation cluster acts independently of the other clusters to perform the functions specific to its role in the overall goal of the system. The only coordination necessary for this scenario is in the access of data which is filtered through the host mainframe computer.

The results of this research will provide criteria which will help a system designer make residency decisions in the following areas:

- Application distribution: Determine where newly conceptualized applications will be most efficiently developed and executed. Many factors will have to be carefully evaluated (e.g. user interface requirements, data requirements, etc.) in order to make such decisions.

- Application migration: Investigate which applications in an already functioning mainframe system should be migrated to the workstation environment and which applications should remain on the host.

- Global functions: Investigate which global functions (e.g. data archival, configuration management, etc.) should be provided by the host and which should be supplied in the workstation environment.

1.3 Document Organization

The results of the research into the determination of application residency will be presented in the following chapters. This document has been broken down into this introductory chapter, one chapter of existing concepts, four functional chapters, and a concluding chapter.

The next chapter concerns the existing concepts and contains a discussion on two methods used to assign software processes to a decentralized distributed system's various processors. The task partitioning and task allocation approaches are used together to effectively assign applications while efficiently utilizing system resources. These models provide the parameters and constraints to be considered when attacking the application residency problem. The methods and considerations proposed in these areas can be modified and expanded to apply to the configuration of one mainframe connected to multiple workstations.

The third chapter contains the hardware elements that need to be considered when looking at overall system performance. The computing power evidenced in executing a solution to a problem is directly related

to the computer hardware being applied. The computing power of the system, however, can be defined as an integrated combination of the hardware performance factors and the software correlation to the hardware factors. This hardware chapter discusses the significance of the performance of the basic computer hardware components in relationship to the computing power. The hardware characteristics to be analyzed are offered; however, the combined mainframe/workstation performance is a conclusion of the combination of the hardware, its organization, and architectures interacting dynamically with the software operating system, and applications programs. Detailed hardware knowledge, based on fair and impartial educated specification information is necessary to separate reality from rumor and manufacturer marketing promises of performance. Hence, Chapter Three will distinguish some hardware components to lead the decision maker to a somewhat objective consideration. These components are provided as the ideal measure of computing power, and are to be taken as a theoretical operational limit. Using the Chapter Three hardware criteria against an available computer characterizes the theoretical maximum amount of computing power accessible. In contrast, as in the case at hand, if hardware criteria is used against a set of computer program requirements the theoretical minimum amount of computing power that is required, in these basic criteria hardware terms, can be determined. With the two separate sets of system knowledge, the residency question can be further addressed.

The fourth chapter covers important concerns in the development of software for operation on a distributed processing system. This topic encompasses the many facets of software engineering: functional decomposition; hierarchical decomposition; reusability of code; and configuration management. It also discusses the importance of an application's interaction with the user, as well as with other applications. The final topics covered in Chapter Four are the timing requirements and response time impacts on program performance. In all of these areas an attempt is made to indicate the importance of maintainability and understandability of software in a cooperating system environment.

In Chapter Five, the considerations associated with the control of various system elements in a distributed computing system are discussed. These elements consist of general resources, interprocess communication and data access. The area of control is an important topic in a distributed processing system because all entities in the system need to cooperate to some degree in order to perform their designated function. In order for all systems to cooperate, there has to be some level of control to coordinate the operation. The selected implementation of an operating system and load distribution and its impact on the applications of a distributed processing system is an important resource control topic covered in this chapter. The other topics discussed in this chapter include the concerns related to interprocess communications and shared data access. Both of these topics concern the acquisition of data quickly and the type of controls necessary to prevent the loss of data and the retrieval of incorrect data.

Chapter Six provides the guidelines which should be applied to the communications requirements to continue the direction for the selection of software residency toward the most efficient hardware support environment. The advent of intelligent workstations for the lowest level of users that have high computing capacities at a relatively lower cost leads to the difficult prospect of coupling a large number of small inexpensive systems to deliver the performance of a large transaction processing system. There are several communications related trade-offs required with coupling N systems, but key problem areas considered are the network transmission speed, coupling protocol overhead, and network errors. The balance between the software functional requirements for network coupling, the network structure available, and the level of overhead interference will be examined.

In the final chapter, conclusions of earlier chapters will be collected to furnish the system manager with a question checklist type of criteria. This criteria will provide guidance in application allocation on a general real-time distributed processing system consisting of a mainframe connected to multiple workstations with numerous multiple functions to perform. Following the list of criteria, methods of using this criteria for determining the distribution and migration of application programs in the system and the residency of global functions will be discussed.

2.0 EXISTING CONCEPTS

There is no clear definition for a distributed computing system. To some, a distributed system is one that allows various users to share resources such as printers, disk drives, and tape units. To others, the processing components must be executing a common application. Another characterization of distributed processing applies to the distribution of processing hardware. A distributed computing system may be composed of a global network of computing facilities, or nodes. A formal definition would include, but not be restricted to, the following characteristics: each node may contain connected computing systems; each computing system may be an interconnection of computers; each computer may be an interconnection of elements; and each element may contain interconnected modules.

The degree of distribution in a system will greatly affect the eventual design and control complexity of the system. At one extreme, systems will be composed of multiple components, each of which operates autonomously. At the other extreme, component-level decisions will be made cooperatively. The degree of cooperation among components may vary from only occasional information exchange to the exchange of information after each decision. The implementation of a system composed of cooperating components, regardless of the degree of this cooperation, immediately complicates the design process. The complication arises in the distribution of functions and data, the communications network architecture design and protocols employed, and the tradeoff between excess hardware and control software.

There were no studies found which specifically tackled the problem of application residency on either a mainframe or an engineering workstation. However, strategies have been developed to assign software processes to a decentralized distributed system's various processors. A considerable number of published works can be found on the use of partitioning and allocation of tasks. These two methods are used together to effectively assign applications while efficiently utilizing system resources. These models provide the parameters and constraints to be considered when attacking the application residency problem. The results of some of those studies will be discussed here. The methods and considerations proposed in these areas can be modified and expanded to apply to the configuration of one mainframe connected to multiple workstations.

Task partitioning is the first step toward assigning tasks to a particular processor in order to maximize resource utilization. The second step is the actual allocation of tasks. These two steps combined can be very helpful in making the decision of whether an application should reside on a workstation or on the mainframe. The next two sections present some of the current approaches to implementing these methods in an effective and reliable manner.

2.1 Task Partitioning

Task partitioning is the process of decomposing software requirements into a set of functional modules and data files (see the discussion on functional decomposition in Chapter Four for more information on possible methods). Once the requirements have been decomposed into modules, all modules are mapped into physical tasks according to intrinsic commonalities, for example, common database-accessing patterns. During partitioning, the sizes of partitions may be constrained in terms of execution time and data storage requirements. Any approach to partitioning must take into account necessary system constraints, including timing requirements, the order of module execution, and limited capacities of different resource types such as CPU throughput, available memory space, and communication link bandwidth. These items are covered in Chapter Three of this document. This partitioning reduces the number of options to be considered during the allocation step (covered in the next section), thereby reducing the overall complexity of computer selection for applications in the distributed-software design problem (Shatz, 1989).

By using task partitioning, the efficiency of resource utilization can be maximized. The objectives of partitioning include minimizing intertask communication, exploiting potential concurrency, and limiting the size of tasks (Shatz, 1989). Some system goals achieved by task partitioning are load balance, minimization of response time, maximization of reliability, and potential for system growth. These objectives will most often need to be considered collectively when trying to assign a task to either a workstation or the mainframe. In order to effectively partition tasks, the necessary response time and potential for concurrent execution needs to be taken into account. Concurrent execution means that two or more modules can execute in parallel if they are partitioned into different tasks.

One problem associated with the partitioning of tasks is the difficulty in measuring its effectiveness. Since partitioning is an earlier design step than allocation, it is difficult to measure the effectiveness of a partition before all processes have been allocated. The other problem is that conflicting partitioning criteria often support the same system requirement.

For example, many distributed real-time applications have critical response time requirements. To meet these requirements, two common partitioning objectives are minimizing intertask communication cost and maximizing potential parallelism. To minimize intertask communication cost, the entire system could be partitioned into just one task and treated as if it were centralized. The intertask communication cost in this case would be zero - there is only one task in the system, so there is no intertask communication. Unfortunately, this strategy does not allow you to exploit any potential parallelism because two computations are eligible to execute in parallel only if they reside in different tasks. The inability to exploit this parallelism may mean that the design does not meet the response time requirement.

At the other extreme, if the system is partitioned so that each module is a task, all potential parallelism can be exploited. But if these tasks are allocated to different processors, all intertask communication becomes remote. In this case, the heavy communication traffic could degrade system response time.

The process of software partitioning is part of the overall software engineering methodology for system development and, as such, it should support the system objectives. One of the most critical system performance goals of a real-time application is to satisfy the response time requirement. Therefore, an important objective of software partitioning should also be established as such. Nonetheless, the response time performance is the product of many interrelated design decisions on issues such as task allocation, node-to-node communication channel bandwidth, node throughput capability, and operating system design. It is extremely difficult to try to measure the quality of software partitioning solutions by means of their potentially achievable response time performance without the tasks being allocated. Before the tasks can be allocated, they are to be defined first through software partitioning. The difficulty in evaluating software partitioning schemes can be circumvented by approaching the problem from another direction - efficient resource utilization. That is, during the process of software partitioning, one can strive to minimize the amount of task dispatching and task communications. The degree of that minimization can appropriately serve as a measurable objective for software partitioning. If the most significant overhead cost is related to task communications, then the above objective is reduced to the minimization of intertask communications cost.

An important factor in minimizing overhead cost is the order of module execution. It can be properly addressed through an understanding of the module precedence relation, which is discussed in detail in the paper "Modeling of Software Partition for Distributed Real-Time Applications" (Huang, 1985). The precedence relation of modules reflects the sequence of module execution. In grouping modules into tasks, it is desirable to maintain the sequence of module execution in order to minimize the overhead expenditure (i.e., delay in response time waiting for a module to execute before executing another module).

To illustrate the module precedence relation, assume that four modules (1, 2, 3, 4) are to be partitioned into two tasks (A, B) with no task containing more than three modules. Let the cost of each module execution be 5 ms, the cost of each task dispatch be 1 ms, the cost of intertask communications (communication between modules in different tasks) be 1 ms for each transmit and each receive operation, and the cost of intratask communication (communication between modules in the same task) be

negligible. Consider the following relationships between the four modules:

- Module 1 passes input to both modules 2 and 3;
- Modules 2 and 3 pass input to module 4; and
- Modules 2 and 3 can execute in parallel.

Consider the following two solutions:

- Solution 1: Task A contains modules 1, 3, and 4
  Task B contains module 2

- Solution 2: Task A contains modules 1, 2, and 3
  Task B contains module 4

Suppose that for these two candidate solutions, both tasks are allocated to the same node and Task A is activated before Task B. The response time performance of these two candidate solutions can be compared as follows.

With solution 1, Task A execution must be temporarily halted, and the task-generated temporary results stored away after module 3 is executed and prior to the initiation of module 4. It will be reactivated after Task B has completed its execution (of module 2) and made the results available to module 4. In this solution, two intertask communications (sending and receiving) are required between module pairs (1,2) and (2,4). Thus, the response time using solution 1 is:

T =	5 ms X 4	+	1 ms X 4	+	1 ms X 3	+	d
	Task Execution		Task Comm.		Task Dispatch		Temporary Storage

= 27 + d (ms)

With solution 2, both defined tasks are completely executable and there will be only one intertask communication required after the completion of Task A. Thus, the response time using solution 2 is:

T =	5 ms X 4	+	1 ms X 2	+	1 ms X 2
	Task Execution		Task Comm.		Task Dispatch

= 24 (ms)

Since solution 2 has less communications requirement and, once executed, both of its tasks can execute to completion, it yields a better response time performance than solution 1. By comparing solution 1 to solution 2, it is noted that by maintaining the order of module execution one can reduce the intertask communication cost, the task scheduling/dispatch cost, the temporary result storage cost, and hence the delay in task completion. Thus, module execution order is an important consideration in software partitioning. Without the module execution relation being

considered, modules within one task were waiting for modules in the other task to complete.

In order to use the module precedence relation to maintain the module execution order, the following definitions and rules must be applied.

- The four possible types of precedence relation are:

  1. One module precedes another
  2. One module succeeds another
  3. One module parallels another
  4. One module precedes as well as succeeds another i.e., neither of these two modules can complete its execution before receiving needed data and information from the other module.

- Two directly connected modules are said to be adjacent neighbors.

- The preceding adjacent neighbors of a module *A* are its adjacent neighbors which precede the module *A*.

- A module is said to be completely executable if it is provided with all needed data and information.

- A module is said to be completely executed if it has made available all data and information to be generated by itself for other modules.

- A module is guaranteed to be completely executable if, and only if, all its preceding adjacent neighbors have been completely executed.

- A module is guaranteed to be completely executable if, and only if, all its preceding modules have been completely executed.

- The hierarchical level of a module in a functional diagram is its logical ordering position relative to other modules in the hierarchy of that functional diagram.

  - A module with no preceding module has a hierarchical level of 1.
  - For a module with a single preceding adjacent neighbor, its hierarchical level is one level higher than its preceding adjacent neighbor.
  - For a module *A* with multiple preceding adjacent neighbors, its hierarchical level is one level higher than its preceding adjacent neighbor which has the highest hierarchical level among all preceding adjacent neighbors of *A*.
  - A pair of preceding as well as succeeding modules have the same hierarchical level and that level is the highest one

found by applying the previous three rules to each one of them.

Keeping the above considerations in mind, the model set forth by Huang will be used as a guideline for task partitioning (Huang, 1985). This model states that the objective of task partitioning is to define an unspecified set of K tasks (for a given set of N modules) so that the software partitioning efficiency is maximized. The task partitioning efficiency is defined as the ratio of total task-direct execution cost to the sum of total task-direct execution cost plus total overhead cost. Hence, the problem is reduced to the minimization of the total overhead cost.

The above objective will be accomplished by observing the following six constraints:

Constraint 1) All N modules considered are included by the K tasks.

Constraint 2) The CPU throughput requirement of any defined task must not exceed either the workstation or the mainframe CPU throughput capability.

Constraint 3) The memory requirement of any defined task must not exceed the local memory space capacity.

Constraint 4) The total execution cost of any defined task cannot be more than the maximally allowed task response time.

Constraint 5) Each pair of modules that precede as well as succeed each other must be included in the same task.

Constraint 6) For a module P_n to be included in a task T_k, one of the following three conditions must be satisfied:

i) Module P_n is the module with the lowest hierarchical level among all modules within the task T_k;
ii) All preceding adjacent neighbors of module P_n are also included in task T_k; or
iii) For each T_k noninclusive module q_n which is a preceding adjacent module of P_n, q_n must precede $P'(T_k)$, which is the module with the lowest hierarchical level in T_k.

If it is known that an individual module within a task will not make available its output before the entire task is completely executed, then one needs to replace the third condition of the precedence relation constraint (6) with:

iii) For a module P_n to be included in T_k, each of its T_k noninclusive preceding adjacent neighbors is

included in a task T_i which contains a preceding module of $P'(T_k)$.

The model presented above uses the maximum software partitioning efficiency as the criterion to determine the quality of the candidate partitioning solution; instead of using the response time performance. The response time performance is the ultimate criterion in evaluating the performance of real-time systems. The reason for this substitution is due to an inability to analytically estimate the resultant response time performance at the software partitioning stage.

The problem of software partitioning can be modeled as one that maximizes the partitioning efficiency while observing the CPU constraint, the memory constraint, the maximally allowed task execution time constraint, and the module execution order constraint. The CPU and memory constraints are implementation dependent. The time constraint on task execution is due to considerations on the response time performance. The constraint on module execution order is a logical one, and it has been properly incorporated into the model by employing module precedence relations.

If there arises a case where modules are executing in a loop, it can lead to an unfortunate solution if Constraint 5 is observed. This constraint would require each module pair with the preceding as well as succeeding precedence relation to be included in the same task. This problem could be alleviated by augmenting the given software partitioning models to include a proper consideration of the precedence relation of modules within a loop. Short of adding that consideration, one suggested approach is to simply ignore the feedback link of a loop, and treat the data and information to be fed back as coming from sources external to the function, or system. This suggestion is inspired by the observation that the feedback link of a loop does not alter the logical execution order of modules within the loop. Under the above arrangement, the presence of feedback data and information can be treated as part of the sufficient condition for a proper task activation and complete task execution, not as part of the necessary condition.

2.2 Task Allocation

Allocation is the step to be taken after task partitioning when assigning tasks to different processors within a distributed processing system. The main difference between partitioning and allocation is that allocation relates characteristics of partitions to characteristics of resources, whereas partitioning looks at commonalities of processing entities with only incidental concern for potential resource characteristics. Allocation binds partitions to physical resources. During the allocation operation, each task defined in the partitioning stage is assigned to one or more system processors. Task allocation complicates distributed-software design because when you assign m tasks onto n processing nodes, there are n^m different possible assignments. In practice, the situation is even worse because you must also consider data allocation and the potential for both data and process replication. Optimal allocation is a problem of exponential complexity (Shatz, 1989).

The key to task allocation is to establish a model in terms of costs and constraints which deal with performance, fault-tolerance, and growth. The goal is to find a solution that minimizes the cost function within the constraints. There are many examples of performance-oriented cost functions, but cost functions that explicitly quantify fault-tolerance and growth properties have not yet emerged. Examples of performance-oriented cost functions are:

- Total interprocessor communication (IPC) cost: Interprocessor communication cost occurs when processes residing in different processors must communicate or when a process must access a remote file. Interprocessor communication cost is a function of the amount of data transferred and of network properties such as topology and link capacity. This topic is discussed further in Chapter Six.

- Total execution and interprocessor communication cost: This is the sum of the total computation cost for each process and the total interprocessor communication cost.

- Completion time: This is the total execution and interprocessor communication cost incurred by that processor whose cost is greater than all other processors.

- Load balancing: This measures how evenly the workload (process execution time) is distributed across the processors. One reason to seek load balancing is to maximize system stability. If a system's workload is unbalanced, there may be a processor responsible for substantially more processing than the other processors. In a sense, this processor represents a weak link ("bottleneck") in the system. (Shatz, 1989)

System constraints which should be considered during the allocation step include the following:

- Limited memory size and processing capacity of each processor

- Dependence of some processes on certain processors, requiring the processes be allocated to those processors

- Limits on the number of processes on all processors (this is one way to approximate load balancing)

The choice of a cost function for a particular system heavily depends on the nature of the application and the underlying hardware. For instance, response time is a critical cost consideration for real-time applications, and minimization of total interprocessor communication cost is more difficult for networks in which processors are geographically dispersed than for local, fully connected networks. For geographically dispersed networks, there is a significant increase in communication time and probability of message loss or corruption.

A fundamental need in task allocation is the development of a strategy for assigning costs to intertask communication. According to one study, the three important parameters that influence task allocation are the accumulative execution time (AET) of each module, intermodule communication (IMC), and precedence relationship (PR) among program modules (Chu, 1989). This study states that the load of a processor consists of AET and IMC. It then proposes an objective function for task allocation that is based on minimizing the load on the most heavily loaded processor ("bottleneck"). A task-allocation algorithm should minimize interprocessor communication by assigning heavily communicating tasks to the same processor unless this would overburden a particular processor and cause a bottleneck.

The task-allocation algorithm set forth in this study describes accumulative execution time (AET) for a module M_j during time interval (t_h, t_{h+1}) as the total execution time incurred for this module during that time interval, i.e.,

$$T_j(t_h, t_{h+1}) = N_j(t_h, t_{h+1})y_j(t_h, t_{h+1})$$

where $N_j(t_h, t_{h+1})$ = number of times module M_j executes during (t_h, t_{h+1}), and $y_j(t_h, t_{h+1})$ = average execution time of M_j during (t_h, t_{h+1}). Both the y_j and the AET, T_j, can be expressed in units of machine language instructions (MLI). Although the execution time of a machine language instruction varies from one instruction to another, based on a given instruction mix, the mean instruction execution time can be used.

IMC is the communication between program tasks through a shared file or message communication on another processor. When a program task on a processor writes to or reads from a shared file on another processor, IMC becomes interprocessor communication (IPC) which requires extra processing and communication overhead. IPC can be reduced by assigning a pair of heavily communicating modules to the same processor.

The precedence relation (PR) among program modules is another important factor that needs to be considered in task allocation. The PR specifies the execution sequence of the modules. Due to PR, a program module cannot be enabled before its predecessor(s) finish executing. The following observations were made regarding the module-size ratio of two consecutive modules and how it affects task response time.

1) Assigning two consecutive modules to a same processor yields good response times if the execution time of the second module is much larger than that of the first;

2) If the second module is much smaller than the first one, the two consecutive modules should be separated and assigned to two different processors.

The task-allocation algorithm proposed by Chu and Lan assumes that:

1) There are J modules, M_1, M_2,...,M_j, and S processors, P_1, P_2,...,P_s;

2) The AET (an average during the peak-load period) for each module M_j, T_j, (j = 1,...J) is given;

3) The IMC (an average during the peak-load period) between each module pair M_i and M_j, IMC_{ij}, (i = 1,...J; j = 1,...J) is given. Each IMC_{ij} can be derived from the communication volume of data sent between the module pair. (Chu, 1984).

The algorithm consists of two phases. Phase I reduces J modules to G groups (G < J) which corresponds to a much smaller allocation task for Phase II. This first phase of grouping can be done with very little computation. Each subgroup generated at the end of Phase I is a set of tasks which will be assigned as a single unit to a processor. In Phase II these groups are assigned to the processors such that the bottleneck (in the most heavily utilized processor) is minimized.

The grouping of modules in Phase I is based on several factors. To reduce IPC, heavily communicating modules may be combined into groups. To do this, communicating module pairs are listed in descending order of the IMC volume. Module pairs with large IMC are considered first.

Next, the PR effects are considered. The decision of whether to group two consecutive modules should be based on the two possibly conflicting factors: IMC volume and the effect of PR. For a module pair (M_i, M_j), the IMC index and the PR index are used to evaluate these conflicting factors. The IMC index indicates the relative IMC size normalized by the average module size in terms of the execution. The PR-index indicates the wait-time ratio of two assignments.

Another factor to be considered is the size of a new group. If the new group, resulting from combining previous subgroups, becomes too large, it would be impossible to obtain a balanced-load assignment during Phase II. Therefore, the concept of processor-load threshold (PL x B) is introduced, where PL is the average processor load and B is a scale constant. If the size of a candidate new group is greater than the threshold, the two subgroups should not be combined.

The two phases of this allocation algorithm generate a minimum-bottleneck assignment. If several assignments yield the same smallest minimum-bottleneck value, then the one with the smallest total processor load should be selected. Chu and Lan propose a function with the objective of minimizing the bottleneck processor load (consisting of IMC and AET) for task allocation. They claim this function generates load-balanced assignments with small IPC.

3.0 HARDWARE SYSTEM PERFORMANCE FACTORS

In the introduction it was pointed out that the computer performance in executing a solution to a problem is a direct result of the computer hardware being applied. The computing power was defined as an integrated combination of the hardware performance factors and the software correlation to the hardware factors. This chapter will highlight the significance of the performance of the basic computer hardware components in their relationship to the overall computing power. The result of this chapter is to provide computer hardware decision criteria to contrast between a mainframe and a networked engineering workstation system when considering the residency of a software application.

The hardware concentration of this chapter is not an indication that there is no tethering of the operational software programs to the hardware specifications. On the contrary, during these hardware discussions operational software considerations will be paramount in all of the final determinations. What is to be realized in this hardware analysis is that a combination of multiple coupled workstations may have some kind of equivalence to the mainframe computer. Further, that engineering workstations have sufficient hardware performance to support the mainframe by executing many of the operational programs independently, off-loading that burden from the mainframe. To what extent these things are true (or even necessary), is very important when determining the overall system computing power. For certain, it is not possible to simply linearly add the individual hardware computer performance metrics for the workstations and the mainframe and assume the total system can support that level of computational need.

Of course, knowing the computational demand, to assess the amount of computing power needed, is the driving issue. The first question is always: how much computing power does the application program, with all of the operational programs interacting, require? The criteria developed in this chapter tries to answer that question and also how much computing power is available at the engineering workstation and at the mainframe.

An additional problem, not covered here, is that a balance must be struck in a combined system of engineering workstations and mainframes. This is because coupling protocol overhead and intersystem interference between programs running on the workstations, but waiting on the mainframe or each other, can slow the execution more than having the basic program directly operating only on a mainframe. Later chapters will examine coupling, controls, data access, communication protocols, and communication systems that influence this relationship and the final estimate of computing power. These factors also influence the resulting criteria principles.

This chapter completes the task of providing basic computer hardware criteria guidelines to estimate the hardware computing power available and the computing power needed by the applications programs. The last section of this chapter will compile the metrics of the various hardware

computer performance elements identified throughout the chapter to build simple models to facilitate this relationship to the residency question.

3.1 Standard Benchmarks for Computer Performance Measurement

The standard method for comparing the performance of two different systems is to execute a selected set of programs in both systems and compare the actual times required for the execution. The ideal case would be to develop the operational program for each computer system that is being considered, and simply compare total execution times. This is obviously an impractical situation, in that the development effort would be overwhelming, and this would still not consider performance of future programs. A more reasonable choice would be to obtain an universal program that can already execute on each of the machines and then compare the execution times to get the relative performance indications.

A program used in this manner is referred to as a benchmark, and analysis of published benchmark execution times (BET) is a reasonable approach at obtaining the actual relative execution times between different systems. Benchmark performance is also reported as the maximum rate of computation (MRC) attained while operating the benchmark program.

Relating the BET or MRC to computing power or even computer hardware performance is yet another problem. Separating the true operational system performance from the performance claims that go along with traditional manufacturer marketing hype is necessary to make correct decisions.

It is always recommended to consider benchmark information on a system before purchase. Rarely is a system purchased without some benchmark reference. Unfortunately, the state of benchmarking is confusing and can often provide inconsistent projections (Dongarra, 1987). Benchmarking difficulties arise as the overall performance is improved through optimized hardware system organization. The advancement of the hardware technology causes most complex architectures to do extremely well on one kind of a benchmark problem, while doing poorly on another seemingly equally valid benchmark program.

The mainframe area of the supercomputer domain shows the greatest swings in performance capabilities between various benchmarks on the same hardware. For example, the CRAY-2 has a top performance conjectural peak MRC at roughly 1951 million floating-point operations per second (MFLOPS). This peak theoretical performance is what the manufacturer guarantees that programs will not exceed, similar to a "speed of light" for a given computer (Dongarra, 1988). However, when solving a system of linear equations with 1000 unknowns, the performance using a tailored algorithm by the manufacturer shows the CRAY-2 actual MRC to be only 346 MFLOPS. The performance is even further degraded when the task is to solve a set of linear equations with only 100 unknowns using a standard software package such as LINPACK, and not fully exploiting the vector capabilities or special hardware features of the machine. With these limitations the CRAY-2 was capable of a MRC performance level of only 21 MFLOPS in an

actual test (Dongarra, 1988). Also note that the difference between the two actual cases is influenced by the fact that the execution speeds have not reached their asymptotic rates in the latter case.

There is significant difference seen in the performance capacities of the CRAY-2 depending on which of the three windows the performance is viewed. In operational applications, similar swings in performance will be seen between the theoretical maximum, the tailored optimized, and the generic situations. Although this problem is more dramatic in the supercomputer environment, this same problem holds true to some degree at all levels of computer hardware.

What can be acquired from this examination is that operational performance of a computer system can not be obtained from timing information presented which reflects only one problem area, solving dense systems of equations using LINPACK programs in a FORTRAN environment, for example. It would further be irrelevant to measure computer systems of unlike architectures, configurations, and manufacturers by comparing performance timing data in situations that are not characteristic or typical of the actual or future applications.

It is possible, however, to include these component measures when the benchmark program is known to be reasonably close to the application need. An obvious improved choice is to select an appropriate benchmark developed along the same nature of the user application requirements. Further, it is not unreasonable to select a "system" of benchmarks that are judged open and fair by an impartial judge such as the set provided by the National Institute of Standards and Technology (formerly the National Bureau of Standards) located in Gaithersburg, Md. Here, some sort of averaging or weighted joint measure would be necessary to arrive at a single MRC if that were the goal. It is still highly probable that the actual overall operational system performance will be poorly estimated relatively between machines until final code implementation is available for evaluation. A reasonable list of criteria would include benchmark data as only a first cut estimate, but because it can easily be obtained many decisions are based on using only this kind of information.

3.2 Independent Hardware Performance Capabilities

Several characteristic computer hardware "elements" exist that limit the theoretical power of any particular machine. These hardware elements can be partitioned into three general groups as follows:

1) Instruction execution speed,
2) Computer organization, and
3) Storage organization.

It is important to note, as was mentioned above, that a particular level of performance from an individual element within one of the groups does not assure the required overall power or even a level of performance of the computer in a particular application situation. Hence, at best, they only provide a rough comparison or guideline between only the hardware

features within computer systems. These are most representative of computing power differences when they are applied between like families of computers. The information becomes less distinctive in value when the comparison shifts to the more generic comparison situations. The addition and combination of these features causes significant swings in the final estimate of computing power.

These three groups listed above will be discussed independently, however there is significant overlap and interaction between the chosen groupings. Regardless, in order to build a set of criteria, there is sufficient published or measurable data for the performance of the elemental building blocks to maintain an independent criteria within each group as it alters the overall computing power.

3.2.1 Instruction Execution Speed

One of the primary measures of hardware performance is the time it takes an individual instruction to be completed. The manufacturer will state the computer's capability in a given number of instructions per second. Since the concern is the high performance end of this discussion, millions of instructions per second or MIPS will be used.

The problem with this measure is that, in reality, each instruction takes a different amount of time depending on the architecture and the microcode of the processor. Note that the reference to instructions at this point is to the number of machine instructions from the hardware instruction set, not the typical higher level language compiled instructions whose timing is additionally software dependent. Also, the instruction execution speed is directly related to the response time (the time it takes to provide the user an output after the "enter" key is stroked by the user). However, actual response times are also related to a multitude of other elements that are both hardware and software generated. Response time also comprises a human-computer interface problem that some designers react to by providing an intermediate response that reports the computer is "working" the problem. This chapter's references to execution speed and performance discounts these intermediate response time designs since they may have no relation to the actual compute time that is necessary in a problem. Response time itself is a topic in following chapters.

The variation in instruction times can be seen in analysis of another fundamental timing element the manufacturers refer to within each of their instruction cycle times. Each instruction can be any number of clock cycles (CC). Depending on its function, the instruction may also require additional number of CCs for time consuming transfers to memory or between registers. Therefore, the faster cycle time stated as the clock frequency (CF) relates to the number of CCs occurring each second. It is a higher resolution, but more sensitive measure commonly applied when comparing similarly equipped machines with the same instruction sets. However, it is less logical to compare the CC time or CF rates to get a measure of relative execution speeds between significantly different machines, considering the broad range of variations possible.

In comparing mainframes or workstations or both, the CC time may still be an important parameter which can quickly guide the selection away from a machine that is advertised as being theoretically too slow for a given need. The opposite conclusion from this equation does not hold true. That is, given what appears to be a CC time that is fast enough for an operational problem, then the computer performance will be sufficient for the need. It would be possible however, given the assembly code of an application program, and the instruction set reference data which includes the number of CC required and the CF, to compute within approximately ten percent the time a sequence of instructions would actually use on the given machine. Even the exact time can be determined by running the instructions on a development system that has an execution vehicle specifically for that purpose. The problem, as it was in application benchmarking, is that these methods rely on the code being completed.

Another measure, one that has been used earlier, is usually applied when there are heavy computational requirements. It is the number of floating point operations that can be completed within a second. This performance factor was referred to as the number of MFLOPS, for millions of floating point operations per second that a computer can demonstrate. This specification is similar to MIPS except that an engineer can usually give a realistic estimation of the total number of calculations necessary without undue effort.

In the situations where only the general nature of the code is known, key instruction execution times can be compared between systems to provide a rough hardware performance correlation. This procedure lacks engineering accuracy and swings very close to the heuristic environment of intuition.

In summary, the elements from the instruction execution speed group to be tied to the general criteria principals are MIPS, and MFLOPS. These two different hardware speed measures will be applied in the criteria set at the end of this chapter. The CF was found to only be of value in comparing machines that are identical except for the CF value itself, and while executing the same instruction sets. Because of this, and the fact that the CF influence is included in the MIPS and MFLOPS values, the CF is normally not considered a general enough parameter to be included when comparing mainframe and engineering workstations.

3.2.2 Computer Organization

The organization of a computer is an important element in the final performance that can be realized. The major differences between computers can be seen in the central processor architecture, the number of processors available, and the methods applied to complete the input and output functions.

3.2.2.1 The Central Processing Unit (CPU) Architecture

The function of the CPU is to execute the instructions that it fetches from main storage. The instructions can be branching, math, loading and storing, etc. In order to execute the instructions, an architecture of

registers, interruption facilities, Arithmetic and Logic Unit (ALU), and even the instruction set itself must be determined. All of these features will affect, in a subjective way, the value of the CPU to the computing power. The goal of this section is to obtain a rating of the CPU based on the features available.

The first component of the CPU to be discussed will be the instruction set method. The instruction set itself has been a recent area of development that provides for further performance divergence between systems. The most significant activity has been related to the extent of the instruction set. A complex instruction set computer (CISC) provides a complete set of complex instructions that were developed to benefit the assembly language programmer. The architecture provides the complex instruction so the programmer does not have to code the set of primitive instructions that correspond to the complex instruction. The complex instruction is a hardware or microcode supported instruction, hence it is also faster in its operation compared to the sequence of fundamental instructions that are equivalent to it. This has obvious benefits, but there is a price to pay in using CISC. That price is increased complexity of the central processor, and increased storage requirements for the microcode. In this way the primitives, or the set of fundamental instructions, are actually impeded in their performance due to the overhead associated with the complex instruction set. Much recent work with the reduced instruction set computer (RISC) has been completed to provide higher final speed of execution. This is possible because today the software development work is done on higher level languages, and the compiled code can be optimized for improved performance using RISC.

The choice between CISC or RISC language sets is also related to the applications selected for the computer. These hardware features of the CPU are, therefore, appropriately related to the software application arena, but will not have a direct intrusion into the power determined other than what is already evident in the other performance measures.

Interruption action (IA) is the second important process considered within the CPU. There are normally four interruption categories allowed by the CPU. They are listed as follows: program interruption (P/I), I/O interruption (I/O/I), hardware interruption (H/I), and operating system interruption (OS/I). Interruption procedures are especially important in real-time applications. However, it is difficult to directly assess the effects the interruption processes have on the other performance factors and the overall computing power. Therefore, as in the choice between a CISC or RISC, the level and capabilities of the interruption process of the CPU will affect the subjective rating of the CPU.

The number and use of registers is characteristic of the CPU architecture and greatly effects the overall performance. The added capability of floating point registers (FPR), along with general purpose registers (GPR) can greatly alter the execution speed when a great deal of high precision computational requirements exist in the application programs. Generally, the more registers accessible, and the greater the capability of the registers, the higher the CPU rating.

The ALU is considered part of the CPU and reflects on its organization. The speed of the ALU is measured in MFLOPS, which was an elemental item discussed in the instruction execution speed group. The relative speeds of the ALU have always been a key difference between mainframe and workstation performance capabilities. The pursuit of swiftness in mathematical operations rather than the pace of program operation is because prevailing applications for workstations rely more and more on mathematical operations. The current engineering workstations can be used to determine the physical stresses on new product designs using finite-element analysis, or perhaps to design and simulate analog circuits with a simulation program. Such analyses demand millions of mathematical operations to complete. Today's typical high-end workstation is often capable of 2 MIPS. This level of operation is commensurate to the IBM Corporation 370/158 mainframe of 1974, at a much reduced cost. However, even when fitted with a standard math coprocessor to improve its floating-point performance, today's most up-to-date workstation can perform math operations at only about 17 percent of that same 1974 IBM Corporation mainframe capacity of 1.5 MFLOPS (Rauch, 1987).

There are generally three varieties of math processors that can be attached to improve the workstation MFLOPS performance. The best known is the off-the-self standard math coprocessor (SMC). This unit acts as an extension to the CPU, and can be added as a simple chip or as an accessory board. The second type is found in graphics applications. Often in this case a dedicated numerics processor (DNP) is added and is embedded in the CPU system architecture. This procedure can increase the performance as much as 100 times that of adding a standard coprocessor. These dedicated math processors must be microcoded, but if the application falls into this category the speed improvement can warrant the difficult development efforts. The third variety is known as general-purpose attached math processors (GPAMP). GPAMPs combine the benefits of using high-level language during design development that is found in a SMC, but nearly attain the same speed as the custom architecture of the DNP. Unlike the DNP however the GPAMP is designed for good performance in multiple applications such as graphics, digital signal processing, and circuit simulation.

It is clear that the metrics associated with adding a special math processor (SMP) within the CPU architecture results in an improved MFLOPS specification and is, therefore, already contained in the performance rating. The nature of this discussion resulted in illumination of the situation, rather than an augmentation to the conclusion. The addition of an SMP is not included in the concept of adding additional central processors to support the solution. Multiprocessor CPU organizations, also called parallel processing, is a separate consideration.

3.2.2.2 Multiprocessors

Another hardware development in computer organizations related to CPU architecture has been the inclusion of multiple processors (MP), in place of a single central processor. These processors operate together in parallel on the problem. Practical parallel processor hardware falls into two major categories. Parallel processors are usually either single-instruction stream, multiple-data stream (SIMD), or multiple-instruction stream, multiple-data stream (MIMD) (Flynn, 1972).

In a somewhat analogous manner, as is the case when comparing many workstations to a single mainframe, adding multiple processors for increased speed performance in a single computer does not add linearly to increase the overall computing power. The gain, if any, that is achieved will depend on many factors. The most difficult to measure is the nature of the problem being solved. Solutions to problems that require manipulation of large data matrices and vectors such as the computation of matrices or eigensystem decomposition can reasonably be expected to be resolved in less time with parallel processing (Huang, 1980). For a more general problem, however, the speedup obtained can be a ratio anywhere from a value much less than one, indicating a slowdown of the process, to a number even greater than the number of processors, depending on the solution's algorithmic situation (Kusmanoff, 1989).

There is a major difference in software development efforts when multi-processors are applied against a problem in an attempt to improve the performance of the system as was described above. The level of effort for true parallel processing requires significant program development efforts by programmers especially trained in the multiprocessor environment. The improvement can still only be accomplished when the application fits appropriately to the environment.

Another extension of multiprocessor hardware exists where the additional processors improve the reliability by having backup processors that simultaneously execute each instruction as it is fetched. IBM refers to these as dyadic processors in reference to their IBM 3081 dual processor capabilities (IBM Corporation, 1986). With a dyadic organization, the primary objective is reliability of the computer, not the improvement in the speed of processing. In many applications, the improved reliability factor warrants the additional cost of multiple processors. In this situation, all of the activity due to having the multiprocessors is transparent to the software development and does not alter development time or effort.

3.2.2.3 Input/Output (I/O) Architecture

The I/O operation is, in reality, distributed among the channel, the control unit, the device, and the CPU. The application requirement for an I/O activity is found within the actual operational program. The methods used from initiation through continuation to termination make up the I/O architecture. I/O considerations associated with the performance

of the computer are the number of channels (NCH) available for I/O, and the channel capacities (CHC).

3.2.3 Storage Organization

The storage organization of a computer plays an important role in how effectively a computer operates, hence the computing power. The storage modes of concern include the memory architecture and peripheral capabilities. The performance measures of storage that are directly involved with computing power are capacity and speed of access to this memory. Indirectly, the word length is also a storage consideration which can influence the speed of operation. The word length, in addition, has a relation to the instruction set length and the real storage capacities. Since these items will be, or have been, discussed separately, word length in itself will not be a topic. However, a direct consideration related to word length is the ability to address memory units smaller than a full word. This will be included within the memory architecture discussion.

3.2.3.1 Memory Architecture

The main memory is one of the major components in any computer. Its primary performance characteristics of capacity and speed certainly relate to the power capabilities of the computer. These characteristics, however, will be seen to have been already incorporated into performance metrics previously established. The relationship of the word length to the storage capacity provides a constraint in the area of absolute storage which in itself can limit the operational program capability if not speed of execution.

3.2.3.1.1 Main Memory Capacity (MMC)

The directly addressable MMC is a constraint associated with the performance of a computer system that also relates to the "size" of the problem. The requirements can be affected by the number of instructions in the operational program, its required database for real-time input use, or the interim output storage expectations. All of these factors interrelate to the operational software demand for memory storage.

The MMC that any computer has is determined by its addressing scheme. For example, a 32-bit computer that has a 32-bit address is capable of directly addressing up to 2^{32} or approximately 4.3×10^9 memory locations. This represents the maximum size of the address space of the computer, and a constraint on the maximum size of memory space that is directly addressable. The actual amount of memory in a system is usually determined by the cost of memory hardware.

3.2.3.1.2 Memory Speed

The speed of the memory can be measured by the time that elapses between the initiation of an operation and the completion of the operation. This measure is referred to as the access time (AT) for the memory. Another scale for memory speed is the memory cycle time (MCT). This is the

minimum time delay that is required between the initiation of two independent memory operations. Commonly, the MCT and AT are not greatly divergent in value, but the MCT is usually slightly longer depending on the actual memory unit implementation. These memory cycle times require synchronization along with the instruction execution times. The differences are not based on available hardware capabilities as much as on the economics of the computer organization. That is, the CPU can usually process instructions faster than compatibly priced hardware memory units. The great expense of using higher priced high speed memories can be mitigated by using a smaller set of high speed memory locations that the CPU sees the slower memory through.

This method is referred to as cache memory, and is based on the fact that execution time is usually spent in a few main routines. The observation is that many instructions in a localized area of the program are repeatedly executed, while other areas are referred to infrequently. The cache memory acts as a high speed buffer between the main memory and the CPU for these often used instructions.

Performance data based on properly coded use of cache memory can often show a significant swing compared to those programs that do not optimize the code for this hardware improvement area. This causes the systems that depend on cache memories to attain higher execution speeds to be more fragile in their general application. This factor needs to be included in the subjective measure of robustness of the computing power.

Because the basic impact of memory speed is seen in the MIPS and MFLOPS evaluations, no additional metric associated with memory speed will be applied to computing power.

3.2.3.1.2.1 Smallest Addressable Memory Unit (SAMU)

Main memory is usually designed to store and fetch full word-length values of memory, although many machines have a capability to address lessor subsets of a memory location based on 4, 8, 16, or integer multiples of 4 bits commonly referred to as a byte of memory. The greater the flexibility in address scheme and the larger the word-length the more substantial the performance factors can be considered. The SAMU will reflect in the robustness value, where the extension of additional memory addressing capabilities allows more robust performance.

3.2.3.1.2.2 Absolute Storage Versus Virtual Storage (VS)

The amount of directly addressable memory is determined by the address structure of the CPU. When the memory requirements have exceeded the capabilities, other techniques have been developed to provide additional memory capacities. The most common is referred to as virtual memory. Naturally, an address specified by the CPU may be an actual physical location in memory. It is also possible that data may be stored in physical locations that require a mapping of the CPU address to locate the physical address. This type of address is referred to as a virtual address (VA). VA is a valuable asset and helps make programs portable.

It is also used to allow bulk storage systems to be added but addressed as if they were in main memory. Using virtual addressing and memory mapping techniques expand the effective memory capacity in a tradeoff of slowing the access times when the mappings occur.

Systems that advertise features of memory extension through memory mapping and VA or an improved speed by applying cache memory techniques require a larger effort to optimize the operational system to take advantage of the improvements. These improvements may effectively show up in the benchmarking activities mentioned in the earlier section. Then again, they may only be productive in special environments of well engineered software development endeavors.

3.3 System Performance Hardware Criteria

It was seen that in this work there is no single number that classifies the computing power. It would be as difficult to characterize computing power with a single number as it is to characterize the overall performance of an automobile with a single number. For a discussion of those that have tried to accomplish a single computing power description goal see the references at the end of this document (Smith, 1988 and Dongarra, Martin, Worlton, 1987). For this study, the power of a computer is seen to be a function of a multitude of interrelated performance considerations. It has been said by experts in this field that no single approach to evaluation addresses the requirements of everyone who needs to measure performance (Dongarra, Martin, Worlton, 1987). Yet some gauge of computing power is necessary to make tradeoff decisions.

This chapter has identified a set of hardware factors which will at least guide the decision maker to a somewhat objective consideration. What this last section will accomplish is to extract from the earlier sections the actual set of hardware criteria that reflects the elements of the three hardware groupings. This criteria set is provided as the best, or the ideal capacity of the computer, and not to be taken as a practical operational reality, but as a theoretical operational limit. Therefore, using this hardware criteria equation against a computer that is available, will attempt to characterize the theoretical maximum amount of computing power that is accessible. On the other hand, when using the criteria against computer program requirements, the theoretical minimum amount of computing power that is needed, in these basic criteria hardware terms, would be determined. The balance of supply against demand will then help to determine software residency possibilities.

Some of the performance values provided are specifically speed related and measurable. They are linked to maximum power in the same manner that an automobile's zero to sixty miles per hour time is linked to power. The analogous computer measurable items that could be listed are elements such as the MIPS, MFLOPS, CC, and MRC.

There are other values that are construction factors such as the inclusion of an overhead camshaft in an automotive engine. These directly associate to the measurable performance elements, hence are already included in the

computing power, but are listed to enhance the subjective input of how the specific levels of performance from the measurable elements are derived. In computer hardware terms, these would be items such as cache memory or multiple processors. They are items that are listed to make the decision maker aware of how the level of performance of the measured elements were obtained. This information provides the sensitivity, or a robustness indication, that reports on the ability of the equipment to maintain the specified performance level under varying environments.

Lastly, some measures of performance contain capacity elements such as a car's interior room or its gas tank limit. Here, an analogous computer hardware component would be associated with the storage organization. In this area, the range of computer performance that can be expected is defined in objective terms. Any of these types of items come into the area of hard constraints from or to the system.

So, holding true to not trying to use a single number to describe computing power, the following sections will provide several combined parameters along with two computed indexes related to the optional considerations that affect quality of the CPU and the robustness that can be expected. These are to be taken together to render a decision on the question of what is the minimum required computing power to meet the needs of the application program. A reverse exercise starting with the hardware available will relate the computing power that is accessible within the workstation or the mainframe computer. Note that this is not the total system solution but only a first step to the solution. Three more chapters follow this one that add considerably more information needed to make the final determination of destination, host or workstation.

3.3.1 Measurable Quantities of Speed

The measurable performance items that are related to the speed of the CPU are the values of MIPS, MFLOPS and the MRC. The CC is a comparable unit of speed but as discussed earlier it will not be included in the final tally because of its limited application and the special constraints required in its application.

Because several quantities have been identified with the speed performance aspect of the computing power, the recommended approach can go into two directions. Either take one of the individual metrics as being the most representative of the speed performance, or take the multiple inputs and build a new composite metric that is representative of all the metrics available. The values used for the MFLOPS and MIPS would be provided from the manufacturer. Neither is always more correct, as each is sensitive to the operational nature of the application program. The different nature of these measures centers on the amount of computation necessary. Because of this, two weighing factors will be defined that can be used to build a hybrid metric of the combined inputs. The weighing factors will be correlated to the nature of the operational program. The fraction of the time estimated that the operational code spends in math computations will be defined by weight, Wc. The fraction of the time the code spends in operations other than mathematical operations will be the

weight, Wo. Wc plus Wo must equal 1.00. The value established for the MRC will follow the guidelines established in the beginning of this chapter and will be in MIPS or MFLOPS depending on the benchmark. For this application, it is recommended to use a geometric average to arrive at the MRC when it is derived from a system of programs. The value computed for the speed of the computer, Sc, in millions of executions per second (MEPS), which is a generic term, will be

$$Sc = ((Wc*MFLOPS + Wo*MIPS)*MRC)^{.5}.$$

This equation provides a speed measure by using a geometric average of the input values in a combined approach including manufacturer specifications and actual execution of code by the CPU.

3.3.2 Measurable Quantities of Capacities and Thresholds

Several quantities such as the MRC, MIPS, or MFLOPS which are extracted from performance tests, published data, or installed hardware are directly related to hard limitation thresholds. That is, if any these capacities are not met, it is equivalent to a system breakdown and the computing power of the system is unable to meet the requirements. These capacities can be associated with the current state of art of the hardware or with the allowed cost of the system. In order to consider these thresholds it is necessary to have the application requirements clearly established to match to the hardware availability. The capacities associated with engineering workstations need to be augmented by the system requirements when distribution requires interaction with application software at other locations.

The MMC is a constraint based on the hardware available. This may be a cost factor or a hardware compatibility factor. In either case, it is a parameter that is provided by the computer hardware and demanded upon by the operational software. The crossover of the demand and constraint is incompatible with successful operation and must be considered a hard limit threshold.

The values of MIPS and MFLOPS as advertised by the manufacturer clearly are also hard limit thresholds, that is, if the demand exceeds the supply the system will fail and the computer power is once again zeroed out. This failure would clearly be theoretically based, however it would still be considered a hard stopping point for decision making.

3.3.3 CPU Rating

Unlike the objective thresholding quantities just addressed, the CPU will receive a soft or slightly arbitrary rating. It will range between 0 and 100 and because of its subjective nature, needs to be considered a soft comparison value. The CPU rating is to be used in conjunction with the earlier established measures of power to help differentiate between different CPU architectures. There are six elements assigned to build the composite CPU rating. A weighing is given to each element, with the subgroupings reflecting partial point value assignments. These weights

are aggregated to attain the final value for the CPU rating. For example, if the computer has CISC (10), all interrupt options (10), only 5 GPRs (3), a DNP (15), two processors (2), 11 I/O channels (10) and a CHC of less than 10 MBPS (5) the CPU rating will be 55. The actual application considerations of the CPU rating will be examined in the final chapter.

ELEMENT	SUBWEIGHT	WEIGHT
1. INSTRUCTION SET		20
CISC	10	
RISC	20	
2. INTERRUPTION ACTION		10
BASIC	2	
P/I	2	
I/O/I	2	
H/I	2	
O/S/I	2	
3. RESISTER CAPABILITIES		10
GPR 1-5	3	
GPR 6-10	5	
GPR 10 OR ABOVE	7	
FPR ANY ADD	3	
4. SPECIAL MATH PROCESSORS		20
SMC	10	
DNP	15	
GPAMP	20	
5. MULTIPROCESSORS		20
$2*LOG_2(P)$ P=NUMBER OF PROCESSORS		
6. I/O CAPABILITY		20
NCH 1-10	5	
NCH 11 OR ABOVE	10	
CHC 1-10 MBPS	5	
CHC 10 MBPS OR ABOVE	10	

3.3.4 Robustness of Computing Power

Several factors were stated to be related to the robustness of the computing power. These factors will all be applied as weights correlated to the difficulty of use, sensitivity to software environment changes, and maturity of the technology. The subjectiveness of these inputs will again make this a soft measure to be applied to the overall computing power of

the applicable weight factors. Several factors are included that have already been inputs to some other component of the computer power. The reentry of the information as a robustness indicator is meant to exhibit the impact on the ability to actually attain the level of computing power. The closer the robustness factor is to one. The higher the probability of the computer system reaching the level of power anticipated or needed. The final robustness factor is the product of the assigned weights if the computer system has the optional factor. For example, if a computer system has cache memory (.99), has virtual storage (.99), and uses multiple processors (.9), its robustness factor would be the product of .99, .99, and .9 or would be equal to .88209. As was the case in the CPU rating above, the actual application of the robustness factor against the residency question will be related in the final chapter.

	FACTOR	WEIGHT
1.	CACHE MEMORY	.99
2.	SAMU	
	BYTE	.9999
	MULTI-BYTE	.999
	WORD	.99
3.	VS	.99
4.	SMP	
	SMC	.999
	GPAMP	.99
	DNP	.9
5.	INSTRUCTION SET	
	RISC	.9
	CISC	.9999
6.	MP	.9

4.0 SOFTWARE DEVELOPMENT

When looking at the placement of application programs in a distributed computing system another area of concern is the development of the operational software. The operational software is the implemented code which is necessary to enact the users requirements. It is also called the application software. There are a large number of concerns affecting the residency of this software which arise during the development of the system. These concerns include:

1) The types of functions which need to be performed;
2) Maintainability of the software across the system;
3) The amount of user interaction;
4) The universal need for the application; and
5) The performance requirements of the application.

In many organizations, the plan for the development, evolution, and maintenance of a system is a separate document from the software requirements since a development plan is considered to be a statement of how the requirements will be carried out. An important component of software development is a methodology that includes management techniques and procedures to assure the success of the project. The current ideas on structured, verifiable, modularized software are all methods used to help attain quality control in software.

In summary, during the development stages of system components there needs to be a methodology used to create a system that is easy to understand, maintainable, consistent, reliable, and verifiable. The areas of system development which will affect these characteristics of a system are structured programming, reusability of code, configuration management, attention to user interface, and program performance.

4.1 Understandability and Maintainability

Standardization is the key to a system that is easy to understand and maintainable. This standardization needs to be applied at the analysis and design stages, as well as at the implementation level. The use of standardized approaches to these areas of a programming system's development will help to ensure that the system is understandable and maintainable. These approaches include the following fundamental components: structured programming (including functional decomposition and hierarchical decomposition), reusability of code, and software configuration management.

4.1.1 Structured Programming

Structured programming is a methodology that lends structure and discipline to the program form, design process, coding, and testing. It is a methodology for constructing hierarchically ordered, modular programs using standardized control constructs.

Although there is much documentation on the necessity for a structured and defined approach to software development, poorly designed and large cumbersome application programs abound in the industry. In a distributed processing system, the necessity for well-structured software is still essential. When there are copies of a poorly designed program running on multiple nodes of a network the inefficiency is proliferated throughout the system.

Structured programming is a method of constructing a program according to a set of rules requiring a strict style format and a standardized control structure. Its common objective is to build high-quality, low-cost software systems. The principles of structured programming seek to improve the management of system development, the process of system development, and the resulting system through the introduction of well-defined procedures, tools, techniques, project controls, and communication mechanisms. They structure the development life cycle into a sequence of step-by-step procedures and use standardization, review, and documentation to provide order and visibility to the process of system development.

According to Martin and McClure (Martin, 1985), the primary objectives of structured computing techniques are as follows:

- Achieve high-quality programs of predictable behavior (reliable)
- Achieve programs that are easily modifiable (maintainable)
- Simplify programs and the program development process (minimize complexity)
- Achieve more predictability and control in the development process (provide disciplined programming methodology)
- Speed up system development (increase programmer productivity)
- Lower the cost of system development.

There are a number of approaches which can be taken to achieve standardization across a distributed system. According to Martin and McClure, the basic approaches are covered by the following principles of structured philosophy (Martin, 1985):

- Principle of Abstraction - To solve a problem, separate the aspects that are tied to a particular reality in order to represent the problem in a simplified, general form.
- Principle of Formality - Follow a rigorous, methodical approach to solve a problem.
- Divide-and-Conquer Concept - Solve a difficult problem by dividing the problem into a set of smaller, independent problems that are easier to understand and to solve.

- Organize the components of a solution into a tree-like hierarchical structure. Then the solution can be understood and constructed level by level, each new level adding more detail.

Structured techniques are a collection of programming methodologies for analysis, design, coding, testing, project management concepts, and documentation tools. Structured programming is accomplished by modular programming, step-wise refinement, levels of abstraction, and top-down and bottom-up programming.

Modular programming is the organization of a program into small, independent units, called modules, whose behavior is governed by a set of rules. Modularization can be applied at different levels of system development. It can be used to separate a problem into systems, a system into programs, and a program into modules.

Stepwise refinement is the process proposed by Wirth for developing a program by performing a sequence of refinement steps (Wirth, 1971). The process begins by defining the basic procedural tasks and data needed to solve the programming problem. This initial definition is at a very high, general level. The process stops when all program tasks are expressed in a form that is directly translatable into the programming language(s).

The levels of abstraction proposed by Dijkstra view a program as divided into conceptual layers or levels (Dahl, 1972). The topmost level represents the program in its most abstract (general) form. All successive levels serve to define the components of this level. In the bottommost level, program components can be easily described in terms of the programming language.

The terms top-down and bottom-up programming refer to adaptations of Wirth's programming by stepwise refinement and Dijkstra's programming by levels of abstraction. When a system is built from the bottom up, the designer creates the components first, makes each component work well, and then fits the components together. When a designer works top down, he first creates the overall structure, defining but not yet building the components. As the design progresses, he fills in the details by building the lower-level components. Top-down and bottom-up design are practiced in many fields of engineering other than programming. On complex projects, a combination of top-down and bottom-up design is usually required.

In order to achieve the primary objectives of structured programming for a distributed system, there are a number of technical objectives which should be met in order to provide a structured design. Before beginning the development stage, a system designer needs to have a clear concept about the systems and functions which need to be performed. Once this information is known, clear diagramming techniques should be used to provide an understandable design of the system's flow. As the design continues, a standard set of control structures should be employed which can be converted into code with minimum effort. When developing software

for any system the complex problems should be decomposed into successively simpler ones. Powerful building blocks and libraries should be used to attain the maximum automation of system design with techniques that make possible the automatic generation of code. An analyst's and programmer's workbench used to maximize help from the computers in achieving objectives is an example of a building block.

An important factor in the understandability and maintainability is the communication which needs to occur during the development of the system. This communication should include the end users as well as the other members of the development team. The end users should be contacted to provide the designers with specifications for a system which is teachable and understandable for the users. Communication among the members of the development team will help to establish rigorous interfaces between separately developed modules and achieve overall consistency across the system. Constant and explicit communication will help to minimize errors and catch those that do occur as early as possible.

The primary goal of all these objectives is to design software which is verifiable and correct while at the same time controlling the complexity of the system. These goals are important in any system, not just a distributed system. They have been discussed here to provide a reference for an approach to providing software for a distributed system which is easy to understand and maintainable.

4.1.1.1 Functional Decomposition

One approach to structured programming is functional decomposition. Most structured design employs a form of functional decomposition. A high-level function is decomposed into lower-level functions; these are decomposed further; and so on. A tree structure shows the decomposition. The term functional decomposition applies to functions rather than data. However, similar diagrams are sometimes drawn for the decomposition of both data and functions.

There are three different categories of functional decomposition. (Martin, 1985) The first and most common type of functional decomposition is a tree structure that relates to function and not to the data that those functions use. The second category shows the data types that are input and output to each function. This can be much more thorough, because if it is handled by computer, the machine can check that the data consumed and produced by each functional node are consistent throughout the entire structure. The third category is still more thorough. It allows only certain types of decomposition, which have to obey precise rules that are defined by mathematical axioms. The resulting structure can then be completely verified to ensure that it is internally consistent and correct.

The specification of generalized and independent functional modules for performing all non-unique application processing is necessary to reduce redundancy of effort and documentation in module development, reduce the redundancy of storage space and execution time during network operation,

and confine the effects of changes to the software to a small number of standardized modules (Schneidewind, 1989).

In a distributed processing system, the components of the system need to be broken down by function. For real-time distributed systems, such decomposition requires consideration of critical timing constraints and may require introduction of special modules such as monitors for module synchronization (MOK, 1984). Once the components have been identified by function, they should be reviewed and any redundant functional modules should be eliminated by developing one global module which performs the function for all applications which need its capability. These global modules could then be put in a library to be accessed by any system application which needed the function. Reusability of code is discussed in more detail in a later section.

4.1.1.2 Hierarchial Decomposition

The modules in a structured program are typically hierarchically ordered. Although hierarchical organization is usually considered an inherent part of modular programming, it is possible to organize a modular program in a nonhierarchical manner. For example, a simple modular program containing only a few modules can be organized sequentially. One module is executed after another. When the last module in the sequence has been executed, program execution stops.

Since sequential organization is not an effective means of controlling program complexity as programs grow in size, a structured program is not sequentially ordered. It should be hierarchically ordered in the following manner:

- The first level of the hierarchy contains one module. This module, called the root module, represents the overall program function at its highest level and describes the viewpoint for the activity.
- The second level of the hierarchy contains modules that further define the function of the root module. At this level, the top-level function is decomposed into several component functions.
- In general, each successive level of modules in the hierarchy provides a more detailed functional description of what the program does.

One hierarchical design for a distributed processing system is based on a hierarchy of abstract machines. At the bottom of the hierarchy is the hardware machine interface. Using the operation provided by the hardware machine, a virtual machine is provided, which is called an extended computer (Faulk, 1988). By using the extended computer interface to hide machine-dependent characteristics, it is intended to make upper level code more portable, abstract from machine idiosyncrasies, provide more readable code, and provide more uniform solutions to machine-dependent coding problems.

The extended computer interface is designed to include only those operations that would have a different machine-dependent implementation should the underlying computer be replaced by one of similar capabilities. The object is to provide the minimal set from which efficient implementations of all useful operations can be constructed. This obviously minimizes the amount of machine-dependent software which would need to be rewritten to transport the system to a different machine.

4.1.2 Reuse of Code/Library

These days, there is great interest in the reuse of software. For example, the Department of Defense is using it as a leading initiative to improve its software technology. In the discussion on structured programming, the idea of reusable code was touched upon. In a distributed processing system where there are clusters of workstations working independently to help achieve the system goal, there are sure to be processes and procedures which need to be performed in more than one area of the system.

A neglected area of great potential in reusability is to design software modules that can be applied to multiple applications. Application development time is significantly reduced and resistance to the ripple effect of future software changes will be maximized by providing a set of server modules which perform all user services other than unique applications functions. Server modules which provide common services are appropriate because certain services (e.g. database management) are not application unique. Rather than having "n" complete sets of application-specific modules, this approach produces only one set of generalized modules in which only the input and output functions are replicated n times.

Reusable code can be produced by applying the modularization and decomposition approaches to software development. As applications are broken down into multiple modules, the designer should be considering the functions which are necessary to other applications in the system. These functions should be classified into independent modules which can be used by other applications.

4.1.3 Configuration Management

Control of software is an important aspect of a distributed system. When there are multiple systems in a cooperating environment attempting to perform one overall task, the integrity of each systems' software must be maintained. Hence, the need for configuration management. Configuration management can be defined as the discipline of identifying the configuration of a system at discrete points in time for purposes of systematically controlling changes to the configuration and maintaining the integrity and traceability of this configuration throughout the system life cycle. Configuration management involves the control of the development and execution of operational software. One of its main goals

is to ensure that only well-tested and reliable software is available to run when the system is performing its intended function.

Software configuration management is one of the disciplines used to attain and maintain product integrity. Achieving some level of product integrity is fundamental to the production of software on a successful basis (successful meaning that the software meets or exceeds the requirements and expectations of the end user). However, it must be noted that successful software development is more than just ensuring that the end product fulfills user technical requirements. It is also necessary to fulfill the requirements in timely fashion and at a reasonable cost.

In the above paragraph the idea of product integrity was introduced. Product integrity is defined to be the intrinsic attributes:

- Which characterize a product that meets user requirements imposed, assumed, presumed or intended during any stage in its life cycle;
- Which facilitate traceability from product conception (as an idea) through all subsequent stages in its life cycle; and
- Which characterize a product that meets specified performance criteria.

In addition, the integrity of a product is diminished if it is not completed on time or within budget. (Bersoff, 1980)

This definition gives particular emphasis to the fulfillment of user requirements, whether specified in advance or not. Frequently, requirements for a product cannot be specified at the beginning of its development cycle. The user may simply not know how to produce a comprehensive specification of his requirements or more likely will omit or misstate some of the requirements; sometimes, the product may even be developed in isolation from the user. Our definition is, therefore, a pragmatic one, demanding, in part, that product integrity be a measure of the fulfillment of the real needs and realistic expectations of the user. It therefore assumes that the developer must carry the prime responsibility for attaining the definition, an Alpine custom split-level house that craftsmen build from the finest materials for a man in a wheelchair does not have product integrity for that user. On the other hand, to a professional mountain climber, the same house may truly posses all the attributes which fulfill the user's expectations, be of superior quality and get the highest marks for product integrity.

Attaining system/software product integrity requires managements's judicious application of many disciplines, including those which are "supporting" in nature versus those which are the "doing" disciplines. The supporting disciplines are particularly important to this discussion because it is through the application of the supporting disciplines that management is able to achieve some checks and balances over the entire

development effort. These supporting disciplines are referred to as product assurance disciplines, and they specifically include:

- Configuration management
- Quality assurance
- Verification and validation
- Test and evaluation

In the distributed computing system the implementation of configuration management is neither more or less important than in a conventional time-sharing system. Configuration management is necessary to maintain a reliable system. In a distributed processing system, reliability is a very important consideration because of dependencies that may be present between various nodes in the system. If a node goes down because of unreliable or non-verified software, it could have a serious impact on the rest of the system.

4.2 Level of User Interaction

There are two views of user interaction to consider when discussing how the level of user interaction affects the residency of an application. One side is the obvious, but important, consideration of whether there is a user interface present in an application. The other side of user interaction is the design of the user interface.

If an application contains user interactive routines, then the response time will be an important consideration when the application is designed. Any application which contains user interactive commands needs to be able to respond to a user's input within a reasonable amount of time to show the user that the input has been accepted. If an application is user interactive, it should reside on the node where it will be executed by the user in order to provide suitable response time. This is a simple guideline to follow if the application has a high-level of interaction (i.e., its main function is dependent upon constant user interaction.)

If, however, an application has only minimal user interaction (i.e., queries for initial inputs) and spends most of its execution time doing data calculations, it would be best to separate the query function of the application from the computation function. By doing this the computational process can then be evaluated individually for its data, memory, and CPU requirements. If the computational process requires more processing power than the workstation is capable of, or if it requires access of system data, it could be executed on the mainframe after being provided with the necessary inputs from the interactive process on the workstation. This would involve a minimum of communication connections to pass the inputs, invoke the computation, and return the results.

The importance in the design of the user interface is that the screen format should be well designed. A well-designed screen format can

increase human processing speed, reduce human errors, and speed computer processing time. A poorly designed screen will have the opposite effect. There are a number of important characteristics to consider during screen design, but only those which are particularly relevant to distributed systems will be discussed. They are:

- Consistency: A system should look, act, and feel the same throughout;
- Design Tradeoffs: Human requirements must always take precedence over machine processing requirements;
- Initiative: For new and inexperienced people, provide a computer-initiated dialogue. For the experienced, permit a human-initiated dialogue.
- Flexibility: A system must be sensitive to the differing needs of its users.

The first characteristic of screen design which is particularly important in a distributed system is design consistency. Design consistency is a common thread that runs throughout these guidelines. It is the cardinal rule of all design activities. Consistency is important because it can reduce requirements for human learning by allowing skills learned in one situation to be transferred to another like it. While any new automated system must impose some learning requirements on its users, it should avoid encumbering productive learning with nonproductive, unnecessary activity. Inconsistencies in design are caused by differences in people. Several designers might each design the same system differently. Inconsistencies also occur when design activities are pressured by time constraints. All too often the solutions in those cases are exceptions that the user must learn to handle. People, however, perceive a system as a single entity. To them it should look, act, and feel similarly throughout. Excess learning requirements become a burden to their achieving and maintaining high performance and can ultimately influence their acceptance of the system. Design consistency is achieved primarily by applying design standards within a common framework. The designer creativity that this stifles (if indeed it does) would seem to be a small price to pay for an effective design.

The second characteristic includes the design tradeoffs which occur when design guidelines conflict with one another or with machine processing requirements. In such conflicts the designer must weigh alternatives and reach a decision based on accuracy, time, cost, and ease-of-use requirements. This leads to the second cardinal rule in user interface development: Human requirements always take precedence over machine processing requirements. It might be easier for the designer to write a program or build a device at the expense of user ease, but this should not be tolerated. This is particularly important in a system with many users that have different levels of experience.

The third system characteristic is initiative. Initiative defines who leads the dialogue between a user and the computer. In a computer-initiated dialogue, the direction is placed in the hands of the system and a person responds to various kinds of prompts provided by the computer. These prompts may take the form of questions, directions, menus of alternatives, or forms to fill in. Computer-initiated dialogues are usually preferred by new users of systems. They rely on our powerful passive vocabulary (words that can be recognized and understood), and they are a learning vehicle, implicitly teaching a system model as one works.

Human-initiated dialogues place the responsibility of direction in the hands of the system's user. The computer becomes a blackboard waiting to be drawn upon. The user provides free-form instructions from memory, either commands or information, and the system responds accordingly. Human-initiated dialogues are often preferred by experienced system users since they permit faster and more efficient interaction. A computer-initiated dialogue tends to slow down and disrupt the more experienced user.

Mixed-initiative dialogues have also been designed. An example of this is labeled function keys on display terminals. The label itself provides a prompt or memory aid but the user must remember when it can be used. Most of the earlier generation computer systems possessed human-initiated dialogue, since this has been the style their designers have been most comfortable with. As a result of the kinds of problems associated with the exposure of computer technology to more nonspecialists, there has been a shift in emphasis to computer-initiated methods. This new emphasis has brought into focus more clearly the problems of this approach for a person who becomes experienced with a system. The result is that today we are beginning to see systems that combine both initiative styles. The needs of both kinds of system users can then be simultaneously satisfied.

Some research has been done to try and determine at what point a person is ready to make the transition from a computer- to human-initiated dialogue (Gilfoil, 1982 and Chafin and Martin, 1980). These studies provided novice system users with a choice of a menu-driven dialogue (computer-initiated) or a command-driven dialogue (human-initiated). In the Gilfoil study participants chose the menu approach to begin with and moved to the command approach after 16-20 hours of experience. At this point they were found to perform better and to be more satisfied with the command dialogue. In the Chafin and Martin study, the transition occurred around 25-50 hours. These numbers, of course, should not be interpreted literally. Many characteristics of the system, task, and using population would substantially influence the results. What is important is the direction these numbers take. They show that it does not take long for new users of a system to start moving from dependent to independent status. An interactive system, to be truly effective, must provide a dual-initiation capability. (Galitz, 1985)

The final characteristic of screen design which pertains to a distributed system is flexibility. Flexibility is a measure of the system's capability to respond to individual differences in people. A truly

flexible system will permit a person to interact with it in a manner commensurate with that person's knowledge, skills, and experience. This characteristic is closely-related to initiative. One kind of flexibility has already been described in the discussion on initiation. A system that permits both human- and computer-initiated dialogues is flexible in that regard. Other areas of flexibility include the display or nondisplay of prompts, permitting defaults, or the creation of special vocabularies. With a flexible system, each person working with such a system can choose the method most comfortable to himself or herself.

Flexibility can have differing levels. At one extreme the user can choose the preferred method and the system will respond accordingly. At the other extreme the system constantly monitors a person's performance (errors, speeds, frequency of use of components, and so on) and modifies itself accordingly. The latter might more appropriately be called an adaptive system.

In order to maintain a consistent user interface, a screen builder which could be used by all software designers in the system would be an appropriate approach. Through the use of such a utility, the look and feel of all screens in the system would be similar.

4.3 Universal Need For an Application

In a large distributed system with many processing units, there are going to be functions and applications which are common to the users at many different sites in the system. The screen builder and configuration management applications are prime examples of such a function. The considerations which need to be evaluated concerning the residency of these global applications include the frequency of use by the users on the system, their necessity during normal operation, the response time requirement, and the data requirement. These factors need to be considered collectively as well as the up-front computing power requirements.

The frequency that an application may be used by each user should be a driving factor in the placement of the application. If the application is used frequently by many users, the response time is going to be an important element in the decision of where to place the application. If a user has a frequent need for a function, the amount of time they will wait for a result will decline as the frequency increases. In this case, it would be more appropriate to place an individual copy at each user's site in order to provide a quick response time. However, if the application has requirements for shared data which is stored on the host/mainframe, it could be possible that the application would perform better on the host/mainframe. In this case the response time may be degraded if multiple users request the same data at the same time. If an application is not used frequently by many users, its placement will have a greater dependency on other elements pertaining to its operation requirements.

Close in hand with the frequency of use of an application is the necessity for the application to be run during real-time operation as compared to development. If an application is to be run during development of the system and not during the operation then the response time requirement is not quite as important. The user will still have a requirement for reasonable response time, but the real-time need for optimum execution time is not present. This situation would tend to lead a system designer to place developmental tools on the mainframe. On the other hand, if an application is to be run during normal operation the response time requirement will be a very important factor and will need to be considered along with the data requirements when trying to determine an application's residency.

As mentioned earlier, a factor in the universal need for an application is the response time required by the users. It has been pointed out in previous chapters that this requirement is an important consideration when trying to determine an application's residency. In regards to the universal need for the application, the response time is again an important consideration. If an application is needed by many of the system users, it will be important to know if they have a real-time need for the application. The response time requirement will be influenced by the demand for the application, the frequency of its execution, and its data requirements. These must all be considered together when looking at the response time demand.

Another important requirement to be considered in the residency of a universal application is its data requirement. If the application, is a system function which needs access to local data then the application would most likely reside on the users local system. If the application is dependent on the shared data stored on the host/mainframe it may be more appropriate for the user to run the application at the host/mainframe. In this case the decision would have to be based in conjunction with the response time requirement and the frequency of need.

It is apparent that the decision for placement of an universal application can be very simple or very complicated. In the simple cases, low usage requirements, developmental applications, non-real-time requirements, and local data requirements sway the decision heavily to either the workstation or the mainframe. In the complicated cases, the response time, data requirements, and frequency factors need to be evaluated collectively so that the most effective solution can be determined.

4.4 Program Performance

One of the most critical requirements put on a system is the timing requirement of the applications which will run on that system. In a system where responses are needed immediately, this is going to be an important criterion for any application which has a real-time need. The available computing power of the candidate systems will have to be able to meet this requirement before the system can even be considered as the location for an application. This determination should not be a problem between the mainframe and an engineering workstation once the computing

power of each has been determined. Computing power is defined here to include all factors that describe both the subjective and objective qualities of the target system. It is thus a measurement of the "success" of the target system, a constraint below which the system must not be allowed to fall. Other factors which must be considered when looking at timing requirements are the effects of other processes executing concurrently on the same processor and overall execution time of the application. When a system designer is contemplating the residency of an application, the effect on each individual program's performance is greatly affected by the number of processes which will be executing concurrently on the same processor.

In many systems, especially those that are embedded in, or connected to, specialized equipment, the real-time performance is essentially a measurement of the success of the system. There are other time constraints which relate events to each other rather than to real time. These relationships would normally involve precedence but might also include information for choosing between competing activities based on some kind of priority system. While detailed decisions on precedence or priority throughout the target system may be left to the designer, there should be a means for including critical constraints in this area in the requirements.

Other factors which characterize the performance of a program are the accuracy and comprehensiveness. The accuracy of the detection and computation of data can be critical. If important data that could be displayed is never made available, or not presented when it could be a determining factor during operation, the target system is performing at a less-than-optimal level. The requirements constrain the eventual design by identifying, at least in general terms, the degree of comprehensiveness desired. Designers can later figure out how to manipulate the data with the quality and human factors constraints.

5.0 CONTROLS

In discussing the assignment of applications to either the host or the workstation, a system designer needs to look at the types of control available in a system to manage the tasks of resource utilization, interprocess communication, and the access of shared data. A distributed processing system gives rise to some problems that do not exist in a centralized system, or that exist in a less complex form. Mainly, in a system which includes a mainframe cooperating with multiple engineering workstations, all nodes need to coordinate to some degree in order to perform their designated functions. This coordination is maintained through the control of system resources, communication, and shared data access.

There are many forms of control in a distributed system. Deciding which type is appropriate for each function in a distributed computing system is difficult. Deciding how distributed control algorithms of different types will interact with each other under one system is even more complex. If these algorithms are implemented in the right combinations then there should be improved performance, reliability, and extensibility-the major potential benefits of distributed systems.

Correct placement of an application program requires analyzing the demands an application program has on system resources, such as memory, CPU time, peripheral hardware. If the program has resource demands that can best be met, or only met, by either an engineering workstation or the mainframe then these resource constraints should be applied in the residency of the application. If an application has a need for communication with other processes then the resource demands of the other processes must also be considered when attempting to place the application. An application's requirement for accessing data shared across the system is another important consideration for assignment of an application to either the mainframe or a workstation. Determining an application's need for resources, communication, and shared data is necessary to evaluate the impact of control methods for these needs on the residency of the application. The following sections discuss methods used to control these elements of a system and the effect these controls have on the residency of an application.

5.1 System Resources

Identification of the methods used to control the general resources on the distributed system being investigated is necessary when attempting to determine the residency of application programs throughout the network. A system designer must assign applications in a manner which best utilizes these resources within the constraints of the methods which have been implemented to control them. When attempting to determine the residency of application programs throughout the network based on the control of resources, the system designer should examine the types of control which are available on a system and specify considerations which need to be made based on the implementation selected for the system in question. The

topics of control which will be discussed here include operating systems and load balancing.

5.1.1 Operating System Control

In a typical computer system, system resources are not directly accessed by users. The users contact the resources through a set of services usually referred to as an operating system. In the operating system arena, functions such as scheduling, deadlock detection, access control, and file servers are candidates for being implemented via distributed control. Consider an individual operating system function to be implemented by "n" distributed replicated entities (controllers). For reliability, it is required that there be no master controller. In other words, each of the controllers is considered equal (democratic) at all times. Furthermore, one of the most demanding requirements is that, in most operating systems, functions must run in real-time with minimum overhead (time sensitive). This requirement eliminates many potential solutions based on mathematical or dynamic programming.

Central to the development of distributed control functions is the notion of what constitutes optimal control. However, such a notion for dynamic, democratic, and time-sensitive functions is not yet well formulated. In fact, this is such a demanding set of requirements that there are no mathematical techniques that are directly applicable. Those techniques which do exist do not address problems such as: inherent delays in the system which cause inaccuracies and eliminate the possibility of immediate response to actions; the necessity for quick decision making; and reliability issues. Furthermore, these theories do not directly deal with stability, an issue that is fundamental to distributed control and reliability.

To have any hope of solving the control problem in a distributed system, either the optimization requirement should be relaxed, more structure should be imposed to the problem, or both. In general, imposing additional structure includes: (1) not only requiring that each controller act sequentially but also to know the action and the result of any action of all previous controllers; (2) various n-step delay approaches; (3) periodic coordination; or (4) using a centralized coordinator. Even with such simplifications, the specification of additional structure does not guarantee that the resulting optimization problem is solvable in practice. Even with this additional structure, the optimization problem can be too complex (and costly) to run functions like scheduling and routing in real time without compromising reliability. Therefore, heuristic methods may be developed which can be run in real time to effectively coordinate distributed controllers in a stable way and do not compromise reliability. Even with a heuristic approach, the delayed effects of the interactions are often not considered. Furthermore, both iterative solutions and keeping entire histories are not practical for most functions in a distributed computing system. With the scheduling problem, there is the added concern that it is difficult, if not impossible, to know the direct system-wide effect of a particular action taken by a controller. For example, assume that controller "i" takes action "a;" and assume that the

net effect of all the actions of all the controllers improve the system. It cannot be assumed that action "a" was a good action, when, in fact, it may have been a bad action dominated by the good actions of other controllers.

Within any distributed computing system, the distribution of control may range from a single fixed control point that makes all operational decisions based on continual observation of the system performance (centralized control) to a fully distributed set of identical control centers cooperating in the decision process governing the operation of the system. The operating system will be a large determination factor in what level of control is implemented on a system.

A recent study at Stanford University relating to distributed computing systems resulted in the development of an operating system, V. The motivation for this study was the growing availability and functionality of relatively low-cost, high-performance computer workstations and local networks (Cheriton, 1988). The basic hypothesis was that an operating system could be developed that managed a cluster of these workstations, providing the resource and information sharing facilities of a conventional single mainframe system but running on this new, potentially more powerful and more economical hardware base.

The design philosophy of the V system was based on three major principles. The first principle was that high-performance communication is the most critical facility for distributed systems. By high performance it is meant that the exchange rate of significant amounts of data across the network should be comparable to that of conventional file access. If the communication rate is slow, it may lead to poor performance and the proliferation of elaborate techniques for dealing with the limited communication facilities. Fast communication allows the system to access files without concern for location, thereby making true network transparency feasible. The second principle was that the protocols, not the software, define the system. In particular, any network node that "speaks" the system protocols can participate, independent of its internal software architecture. Thus, the challenge was to design protocols which would help attain the performance, functionality, reliability and security required by the system goal. The final principle was that a relatively small operating system kernel can implement the basic protocols and services, providing a simple network-transparent process, address space and communication model. The rest of the system can then be implemented at the process level in a machine and network independent fashion.

A cooperating computer system requires a cohesive control structure to bind its components and a local operating system for each of its node computers (workstations and mainframe) to execute. Knowing what type of controls an operating system in a distributed processing system has implemented is important to the allocation of applications between nodes. If the operating system is too localized, unable to provide the commands needed for communication between processors (like DOS) then there will be very poor interaction and no overall control. If the operating system is designed for multi-systems (like V) then the users will have better

control and function. In a system of multiple engineering workstations connected to a mainframe computer containing a centralized database, there needs to be a degree of decentralized control for communication and resource allocation purposes. There could also be centralized control at the mainframe level for the access of shared data.

5.1.2 Load Distribution

Most load distributing algorithms can be categorized as following one of two archetypical strategies - load sharing (LS) and load balancing (LB) (Krueger, 1988). LS attempts to conserve the ability of the system to perform work by assuring that no node within the system is idle while there exists a demand for service anywhere in the system. LB goes a step further by striving to equalize the entire workload among all of the nodes. LB can further be characterized by the static or dynamic nature of its control. Static control only deals with the initial placement of a process. Dynamic control employs a migration component capable of transferring a process once it has already started to execute.

In distributed computing systems, these algorithms have been developed to improve the performance of the system (e.g. to minimize the mean response time of a job) by efficiently utilizing the computing power of the entire system. The availability of facilities which allow a user on one workstation in a distributed system to execute a job on another workstation in the system has led to an effective means of maximizing the computing power within a distributed system. Through the use of these remote execution facilities, the user of a workstation-based distributed system can transfer jobs from heavily loaded nodes to inactive or less-busy nodes.

Recently, this method was applied to find the prime factors of a 100-digit number. It had been theorized by computer scientists that a single computer doing a million calculations per second would have needed 25 years to solve the problem. Even a state-of-the-art supercomputer such as the Cray would need about 10 months of constant computing. However, the number was factored in just 26 days by using the idle time of 400 computers in the United States, Europe, and Australia.

Since workstations are typically allocated as private resources for the user who controls access to them, the load distribution method aims to maximize the utilization of processors with as little interference as possible between the jobs it schedules and the activities of the users who own the processors. This is an important concern for the users of workstations connected in a distributed processing system. If a user at a workstation has a time-critical application which needs to execute, that user does not want an application from another node trying to execute in the background at the same time. The capability to control when the processor is free to execute remote processes would be a desirable feature for the users of workstations.

Although a communication delay must be incurred by transferring a job from one node to another, the performance of a distributed computer system

can generally be improved by an effective load balancing policy. An example of this type of load distribution is used by the Condor scheduling system (Litzkow, 1988). Condor operates in a workstation environment. This system identifies idle processors in the network and schedules background jobs on them. When the user of a workstation resumes activity at a station, Condor checkpoints the remote job running on the station and transfers it to another workstation. The system guarantees that the job will eventually complete, and that very little, if any, work will be performed more than once.

Obviously, load distribution, both LS and LB, involves coordination and cooperation between the various nodes of the distributed system. Hence, there is always communication delay of transferring a job. If a distributed system does not implement some method of load distribution, applications can only execute locally. In this case, however, the assigned processor must be capable of providing the response time required by the applications.

In summary, the questions which need to be answered about a system with regards to load distribution are is the system capable of load distribution and if so what type is implemented: LS or LB (static or dynamic)? If a system does have a load distribution method active then are there any background applications which cannot be controlled by the load distribution because of special resource demands which would eliminate the application being passed off to another processor? If there is not a load distribution plan, then all resource demands will need to be considered in the static allocation of background applications to each processor.

5.2 Interprocess Communication

The considerations of interprocess communication control are discussed in this section. A computer network consists of a collection of host computers connected by a communication subnet. The subnet physically transfers messages between nodes of the network. Most networks are designed as a layered system which allows a layer at one site to communicate with "peer" layers at other sites. The rules governing this communication are called protocols. The complete set of layers and protocols in a computer network is known as a network architecture.

Functions in the subnet such as access control, routing, and congestion control are good candidates for being implemented with distributed control. Routing is the decision process which determines the path a message follows in passing from its source to its destination. Some routing schemes are completely fixed; others contain fixed alternate paths where the alternative is chosen only on failures. These non-adaptive schemes are too limited and do not fully utilize distributed control. Adaptive routing schemes modify routines based on changing traffic patterns. Adaptive routing schemes may be centralized where a routing control center calculates good paths and then distributes these paths to the individual hosts of the network in some periodic fashion. Again, this is not an effective use of distributed control.

Routing algorithms which exhibit distributed control typically contain *n* copies of the algorithm (one at each communication processor). Information is exchanged among communication processors periodically or asynchronously as a result of some noticeable change in traffic. The information exchanged varies depending on the measurement used by the algorithm (e.g. the number of hops, an estimate of delay to the destination, or buffer lengths). Each copy of the routing algorithm uses the exchanged (out-of-date) information in making routing decisions. Such algorithms have the potential for good performance and reliability because the distributed control can operate in the presence of failures and quickly adapt to changing traffic patterns. On the other hand, several new problems arise in such algorithms. If the algorithm is not careful, then phenomena known as ping-ponging (message looping) and poor reaction to "bad news" might occur. These problems are essentially stability problems and affect the reliability of systems.

Another type of distributed routing algorithm is based on "n" spanning trees being maintained, one to each site of the network. Each spanning tree is largely independent of the other trees so this is not a highly cooperative type of distributed control. Such an approach does have a number of advantages such as guaranteeing that there will be no looping of messages, thereby solving the stability problem. There is also another degree of reliability provided because each site has its own tree; but additions to the algorithm are needed to further increase the reliability so that an individual failed site can be bypassed (i.e., alternative paths should be provided).

When too many messages are in the subnet, performance degrades. Such a situation is called congestion. In fact, depending on the subnet protocols, it may happen that at high traffic, performance collapses completely, and almost no packets are delivered. This is another form of stability problem and, hence, is also related to reliability. Solutions include preallocation of buffers and performing message discarding only when there is congestion.

A particularly interesting distributed control algorithm for congestion control is called "isarithmic congestion control" (Stankovic, 1985). In this scheme, a set of permits circulate around the subnet, and a set of permits are fixed at each host. Whenever a communication processor wants to transmit a message, it must first acquire a permit, either one assigned to that site (and not being used) or a circulating permit. When a destination communication processor removes a message from the subnet, it regenerates the permit. Stationary permits are considered free upon message acknowledgement. This scheme limits the number of messages in the subnet to some maximum given by the number of permits in the system. Although this scheme enhances reliability in one or more ways, there are still reliability issues which are left untreated. These issues could be addressed if this scheme included what actions to perform in the case of lost permits or downed sites.

An unfortunate characteristic of message communication is that it is possible for a message to become lost while passing through the network. This might require users to program explicit acknowledgment schemes into the application-level procedures, which then adds to the communication cost by both time and lost acknowledgements. Furthermore, because of the distributed nature of most systems that use messages, messages may be received in an order different from that transmitted, and messages may arrive at the destination buffer from different sources. It is also possible for a sender to issue a burst of messages which can not be addressed quickly enough before the sender times out. Because of these problems it is necessary to implement reliable interprocess communication. The protocols established for system communication are necessary in any applications which plan to incorporate interprocess communication. In order for one application to communicate with another application, there needs to be a common set of procedures used to establish a connection between the two applications and transfer the information between the sender and the receiver.

The control of interprocessor communication can be characterized in roughly three categories - master/slave, dialogue, and mailing system. (Stankovic, 1985) In the master/slave approach the slave is available at all times, the master does not ask permission, and the slave notifies the master when it has completed its task. This kind of IPC can be accomplished in hardware by interrupt signals and in software by procedure calls. This is a tightly coupled IPC construct, and its use is often very efficient because the master does not have to wait for permission from the slave. This construct, as normally implemented, can block for long periods of time if, for example, the slave crashes after it is called.

The dialogue approach requires the user to acquire permission for using another processor. The activity of the other processor is triggered by the user's request, but controlled by the processor, and the connection between the user and the service processor is established temporarily.

The dialogue approach can be implemented by procedure calls - i.e., a series of calls back and forth between user and facility with the appropriate checks and parameters. In many instances of dialogue communication, there is no specific need for the message. They are just checks and acknowledgements. However, if the two conversing parties are physically on different processors, or if there are many conversing parties, the message mechanism is more natural and better suited to these tasks. The reasons for this arise from data flow considerations. First, the sender and receiver are on different nodes and, therefore, difficulties arise in sharing global variables and referencing environments. Second, with multiple conversing parties, the buffering of messages may become necessary. Reliability issues arise at a number of places in the dialogue approach.

The third category of control is the mailing system approach. In this approach, information is not sent directly from source to destination because there are intermediate stops. The information is sent whether or not it can be processed immediately, or is even capable of ever being

processed. Typically senders (receivers) have some kind of buffer into which (from which) information is placed (removed). An important characteristic of messages is that the number of messages sent can vary with time; that is, senders can operate in burst mode where a number of messages are sent in quick succession and none are sent for a period of time. Furthermore, receivers may not be able to process each message right away; hence, the need for buffers. The semantics of the procedure call do not support a mailing system. Users would be required to implement the mail features themselves. The message is the communication method developed for implementing mailing systems, but the level of reliability associated with messages is highly variable depending on the implementation.

The things a system designer needs to investigate when allocating applications to particular processors in a network are the amount of interprocess communication between an application and other processes, the number of other processes which communicate with the application, and the importance of response time in the application's execution. If an application has a high degree of interprocess communication, it should reside on the same node with other applications it communicates with in order to reduce the response time which would be dependent on the communication delays. If this requirement would overload a single processor then the least involved processes should be moved to other processors which can meet their computational needs. If the communication control software does not distinguish between local and remote communication then the difference in response time may not be noticeable, but the reliability of local communication would be higher. Chapter Six covers the factors associated with delays caused by the use of a communication network.

5.3 Shared Data Access

A primary strength of a centralized system using the time-sharing technology of a mainframe computer came from the ability to share stored information among all the users of a system. A common problem on a distributed processing system is the need for different processors to share the same data. Where and how this data should reside on the network and what the process should be for controlling access to it are important questions to be answered when allocating applications in a distributed processing system.

In many cases it is necessary for two or more processes to have simultaneous access to the same file. However, a process that modifies a file cannot share with any other process access to that file at the same time. In order to provide users with this flexibility, a system must provide two methods of access to every file, either shared or exclusive. The user specifies which method of access (shared or exclusive) is desired when the file is requested. If the process has not been granted exclusive access to a file, then a request for shared access to that file by any process can be granted. In the same manner, if no process has been granted either shared or exclusive access to a file, then a request for exclusive access to that file by any process can be granted. This

protocol provides multiple readers (shared) or one writer (exclusive) with access to a specific file.

A deadlock exists whenever two or more processes vying for the same resource reach an impasse. That is, neither process trying to access the resource may proceed until they are granted access. When a deadlock occurs, if a job is aborted, the resulting partially completed process often represents an inconvenience. The user whose process aborted must, in many cases, reconstruct partially altered files. For this reason, deadlock is an important consideration in the design of operating systems even though in practice it seldom occurs.

Data sharing and data partitioning are two different approaches for coupling multiple systems which need to access the same data. In the data sharing approach, all the processors have direct access to the common data, and accesses are coordinated by appropriate protocols. In the data partitioning approach, the common data is partitioned among different processors and a function shipping mechanism is used to access data on remote systems. In a paper by Dias, Iyer, Robinson, and Yu (Dias, 1989) the data sharing approach and methods of enhancing its performance are discussed. Using this approach, all processors in the distributed processing system have access to common data at the disk level. In some partitioned approaches data is replicated on different processors. The reason for this replication is to improve the reliability of the system. If a processor containing necessary data failed, then the replicated data on another processor could still be accessed and the system would be able to continue functioning. The replication of data would mean that there needs to be a control mechanism for insuring that all instances of data are simultaneously updated. If this type of control is not present then synchronization problems could occur when the system attempted to switch-over to the backup data because the data would not be in the same state.

There has been quite a bit of research done in the area of distributed database system architecture. One of the main research issues in distributed databases has been concurrency control. In order to manage simultaneous access among transactions running on different systems, global concurrency control of the data is required. Various algorithms for concurrency control have appeared, including some based on distributed control. In one such algorithm, integrity of the database is maintained by distributed controllers in the presence of concurrent users. The distributed controllers must somehow cooperate to achieve a system-wide objective of good performance subject to the data integrity constraint. This cooperation is achieved by the combined principles of atomic actions and unique time stamps. Another class of algorithms have also been shown to achieve this same cooperation based on two-phase locking and atomic actions. However, many of these solutions are not robust, i.e., they must block on failures.

In the scenario of a cooperating mainframe host/engineering workstation environment, the mainframe host contains a centralized database of global data needed by each of the connected engineering workstations. The closest discussion found on this type of environment discussed load

sharing in a hybrid distributed-centralized database system (Ciciani, 1988). With this type of system some transactions run at (geographically) distributed systems, and other transactions at a central computing complex. Such a system can provide the advantage of distributed systems for transactions that refer principally to local data, and also provide the advantage of centralized systems for transactions that access a lot of non-local data. In a fully centralized system, where user terminals are connected by a network to the central computing complex, all transaction input messages are shipped to the central system where the transaction is processed, and output messages are sent back to the terminal; hence the centralized system does not make use of the possibility of a local data reference. The distribution of a database may range from a single file or file system with a file directory maintained as a single copy in a central storage medium (totally centralized data base), to a multiple file system with replication of both files and directory maintained across several storage mediums.

Another method by which a workstation can achieve control of shared information is by including a file system that is capable of accessing remote files via a local area network attachment. These file systems are commonly known as distributed file systems. There are three major types of distributed file systems: remote-disk systems, block-level-access systems, and file-level-access systems. A remote-disk system implements a method of sharing disk drives among a number of workstations by communicating disk I/O requests via a local area network. A block-level-access system implements a method of sharing disk files among a number of workstations by communicating I/O requests for file blocks via a local area network. This method allows more than one user to read or write to a file at a time. A file-level-access system implements a method of sharing files among a number of workstations by transferring entire files via a local area network.

A combination of these approaches to shared data can be used to exploit the advantages of each. A remote-disk system eliminates the need for a local disk, and provides shared access to immutable data. This eliminates the need for a local disk, reduces the cost, size, and power consumption of a workstation. Block-level-access systems permit mutable blocks to be shared among workstations. The systems containing the shared files are referred to as file servers. These servers implement directory systems that can not be confused by concurrent access, and thus it is safe for multiple users to create, delete, and mutate blocks of files. High performance networks and file-servers must be used with a block-level-access system because multiple server requests are required to access an entire file. Block-level-access systems can also be used to eliminate the need for a local disk. File-level-access systems permit mutable files to be shared among workstations. Because entire files are retrieved at once fewer access to file servers are necessary. However, when individual blocks of files change rapidly, it is not practical to fetch entire files.

Other controls which are necessary to maintain the reliability of the shared data and which have an impact on the allocation of applications to either an engineering workstation or the mainframe include data location,

deadlock prevention and detection, integrity and consistency in multiple copy data bases, fault tolerance and error recovery, query optimization, data translation among heterogeneous data bases with different data structures, communication protocols for distributed data bases, performance monitoring and measurement, and security and privacy issues. If a task has to retrieve large amounts of shared data at different times during its execution then that task needs to reside on the same node as the data in order to cutdown on response time.

6.0 NETWORKING DELAYS

Chapter Six contributes the last supplementary guidelines for the selection of the software residency in a merged mainframe/engineering workstation environment. The final parameters that are included involve the consequences that arise because a distributed system demands the use of an external communication network. Whenever the workstation residency is considered, then the added delays of a complex communication network may be required also. This is not a subtle difference, because if the data and operational programs are all co-resident, and only an output display needs to be relayed to the user, then the communication demands may be quite small. To know the true communication demands, and then calculate or measure the impact is the focus of this chapter.

There are three basic delays that result when communication is extended between separate hardware entities over a network. The first is the delay associated with the difference in the slower transmission speed of the network versus the speed of accessing shared storage in a non-distributed system. The second network originated delay is the price paid for the coupling protocol overhead processing which is necessary to achieve network communication. A third delay or slowdown is connected with the intersystem interference brought about with conflicts between simultaneous network system communication activities. To some extent, this was already discussed in Chapter Five relating to the control activities that are necessary when a database is shared across network boundaries and this particular effect will not be revisited here. However, the congestion slowdown of the network due to attempts at simultaneous access is a major communication problem and will be the concluding topic on network delays.

In actual situations, it is necessary to size the requirements against the dimensions of the capabilities to decide the impact the communication network has on the operational performance. This relates directly to the question of the residence of the operational software.

6.1 Demand For a Network

Two basic arguments are always present. Given a mainframe computer that has the computing power necessary, as defined by the performance factors in Chapter Three, any one of the operational programs would typically operate faster if it could rely exclusively on the mainframe for its computing demands. On the other hand, given engineering workstations with enough power to complete the application execution within the constraints imposed, and given enough workstations, there would be little requirement for a mainframe.

In the former case, the workstations perform as simple input and output devices requiring only output display activities and request inputs. There is an obvious pitfall when such a plan is followed for all of the operational programs. It is that the mainframe could fall short of meeting the requirements of one or more of the individual operational

computer programs while trying to simultaneously meet the accumulated demands and process all of them.

In the latter case, the assumption limits the scientific computational capacities of the operational software and the inter-application needs. Hence, unless the operational applications are totally uncoupled, a communication network is necessary to exchange information between the workstations.

The two extreme ranges are rare; therefore, this study has assumed neither of these limits is ever the actual case. That is, it is assumed to be necessary and possible to off-load some portion of the mainframe computing demands to an engineering workstation. It is also assumed that there are application demands that require a mainframe level of computing power. In any case, it is also reasonable to make the assumption that there is some coupling necessary as is found in any cooperative work and group decision making activities which then demand system communications. The contribution of this chapter is to fold into computing power the information associated with the impact of the network communications between the mainframe and the engineering workstations while trying to balance the workload to maximize the overall available computing power.

6.2 Network Transmission Speed

In the last ten years network communication has gone from transmission speeds at the physical interface of 110 bps and 9600 bps to fiber networks that support 10 Mbps and 100 Mbps channels. The simple fact that high speed circuits for LANs are now available does not mean that they exist or are reasonable on all applications. Systems that use the lower speed rates because of cost tradeoffs or dependence on older telecommunication systems will find that the data speed is the bottleneck causing the greatest delay of the three delays possible because of networking. On the other hand, the newest high speed optic links operate as fast as the software procedures can transmit the output data or receive the input data, hence do not enter into the performance slowdown at all.

The hardware performance portion of the communication delay can be reasonably estimated when one knows the message lengths and the number of messages required by the operational programs. Merging these message requirements with the network channel structure can provide the timing estimates to compare with the application program parameters when designing a network. If the network is already available, this procedure will relate the potential of the network to meet the demand of the software requirements. The key parameter being the capacity of the channel versus the capacity demand of the operational system.

6.3 Protocols and Communication Procedures

There are a great many software activities required whenever communication is deemed necessary. There are the obvious software interactions that can be clearly identified within the application code. These are the basic primitives that make the communication setups or calls. These simple

operations consists of only a few instructions but are the collision point of the operational program and the communication network.

A great deal of technology movement and study has occurred in the last ten years to develop a standardized way of communicating over a computer network. The most common description, based on seven layers of subsystems which have been developed by the International Organization for Standardization (ISO), is called the Open System Interconnection (OSI) reference model. The International Telegraph and Telephone Consultative Committee (CCITT) consistent with the ISO model has published its standard X.200, Reference Model of Open Systems Interconnection for CCITT Applications. Each of the seven layers has an interface that services the layer above it, and a peer protocol that has procedures between peer entities with the destination and origination points at the same subdivision level. Each layer function adds value to the services of the entire set of lower layers until the highest layer has available to it the complete set of services required by the application. The layering of functions has influence on the communication throughput in two ways.

The first is by the extension of the basic message unit length due to the protocol header-trailer information that is necessarily added at each level. Normally, the basic message length can be a variable consideration which allows change in the message overhead to message content relationship. However, limits on maximum length of the messages are imposed by the protocols, by the nature of the exchange, the quality of network service, and the buffer space. Studies have shown that a measurable percentage of the communication bandwidth available is used to service these protocol overhead needs. This is an applications program dependent consequence and the cost can be approximated as a percentage of the available communication bandwidth knowing the protocols selected for implementation, and the types and amount of message traffic required.

The second effect is the processing time required to service each layer as the message is passed down and up through the layered entities. This is related to not only the protocol selected but the operating system relationship to the communication activities. Access to memory, how often the message needs to be touched, computation of checksums, and interruption procedures all affect the delay from network communication and are functions of the implementation method within the operating system.

Often the lower three layers of protocol, the physical, link, and network, are implemented in silicon (firmware and hardware) and the central processor can even be supported by its own separate communication processor. Then again, there are networked systems that do not off-load any of the communication processing, causing considerable impact on each communication demand. Processing delays associated with network communication can be estimated based on the number of instructions used by analyzing the code and knowing the Chapter Three hardware performance factors. It is also possible to use network benchmarking procedures that assist in predicting performance in various environments.

6.4 Network Error and Congestion

Up to this point all of the communication considerations have treated the network as if it were error free, and there were no simultaneous user communication network requests. Contrary to this, errors can be expected to occur in transmission and multiple (if not all) users may try to communicate at the same instant. Both of these problems will cause additional delays associated with dead time consumed in queues and duplicated message retransmission requirements. The extreme limit of this delay would be a locked up network, with no response possible without restarting the system.

6.4.1 Transmission Errors

The engineering analysis of a transmission system contains a large number of parameters of concern. Primarily these include, but are not limited to: frequency response, interchannel modulation, idle channel noise, received signal level, and delay response. These parameters are all physical characteristics which can cause an error in the transmission of the signal if they fall out of system tolerance. The number of errors resulting from these problems and any other problem are grouped into a single digital communication performance measure. That parameter of performance of the network due to the expected errors is the bit error rate (BER) of the communication system. The BER that can be expected is dependent on all of the hardware factors of the system.

A longer distant network like a metropolitan area network (MAN) or a wide area network (WAN) is more prone to have errors, hence high BER, due to less reliable communication systems and varying conditions of the environment. In the case of a LAN the BER is usually very small and if fiber optic transmission systems are used, the BER is nearly non existent in a properly maintained and installed system.

Within the layers of protocol there are error checking and correction techniques that shield the user from being aware of problems that are occurring. However, a troublesome communication link can cause a significant slowdown in performance by limiting the throughput of the system. Retransmission of messages in error increases the load on the system. If the BER is high enough, communication is no longer possible. At this point all users will be well aware of the situation and once again experience a locked up (non-communicating) network.

6.4.2 Communication Network Congestion

Any computer communication network has a finite communication capacity. This is true for a LAN, a MAN, a WAN, or even the common telephone network. The basic problem that causes congestion is the same in all cases. With the telephone system network, there exists a capability for every user to make a connection, but not all users can do so simultaneously. In computer communication networks a similar situation occurs, along with additional options to improve the overall network connectivity or usage. In both cases however, when the demand approaches

the capacity, the communication is hampered by not being available and this kind of slow down is termed a congested network.

6.4.2.1 Telephone Networks

In the case of the telephone system, a system of switches, trunk lines, and statistical factors control the size of the system to optimize the cost versus performance of the system. The limits of the system are associated with two factors, what connections are trying to be made at a point in time, and the number of simultaneous connections allowed throughout the system. In the first case, if a destination user is busy when a connection is trying to be made to it, then the connection fails. This is signaled to the originating user with a busy signal of slow buzzes. In the second case, systems are sized by average loads, and average peak loads, assuming that all users will not be making demands on the communication system at the same time. When the total load is too great, the local switch returns a busy signal of fast buzzes to the originator, even if the final destination may be available at that point in time. In the case of the telephone system a hard-wire dedicated path must be established, end-to-end, before the two users can communicate. Further, the dedicated path must remain established during the entire exchange, and is taken down to terminate. Trying to making a long distance phone call on Mother's day is an example of a congested telephone network and its delaying effects on communication.

6.4.2.2 Computer Communication Networks

Computer communication has some of the same constraints as the telephone system, but also has some additional options. Computer communication networks can be designed to work with a connection or a connectionless mode.

In the connection mode the network functions are similar to the telephone system requiring the end user to be available to set up a logical rather than physical connection between the users. This logical connection is an end user to end user path that remains current during the communication dialogue and consists of reserved buffers and logical channels. It differs from having to have a fixed dedicated hard wire path end-to-end by allowing the path to change during the conversation internally on the network. Additionally, the hard-wired path or paths will be supporting many other logical paths during each other connection's idle block time. A computer communication network is the same as a telephone network in that the end users can not communicate information until the connection is completely established end-to-end. Also, the responsibility of error free communication lies with the end users and not the network.

The connectionless mode allows a user to release a message to the network and fastens responsibility for the message to the network. The network gives an assurance to the sender that it won't give up until the message is delivered (although this is not always the case). This is analogous to the U.S. Mail system where the letter (message) is addressed to the destination and released to the post office (network) which promises

delivery of the originator's letter to the destination address. There is never an end-to-end connection established and the message can go anywhere necessary before it reaches the final destination. The limits of a network operating in the connectionless mode are associated with buffer storage space available and the number of messages existing at any one time within the network.

Communication networks, connection or connectionless, are also designed for average peak loads in a trade off of cost for capacity. Even costly designed networks, after a brief period of growth may find that the demand equals or exceeds the capacity of the network. When this first occurs there is a slow down in the network that is manifested in a slower response time of the operational program. This congestion can cause delays, lost messages, and eventually if the load continues to grow, a total lock up of the network.

In order to make the decision for residence of an operational program, the communication needs should be clearly defined so that the risk of failure of the individual program and the failure of supporting communication network can both be assessed. The overall supporting communication network demand is, of course, based on the aggregate of each of the operational program demands. Therefore, system wide control must be imparted to protect the resource of the network as additional demands arise. What individually appears to be correct can, in the total picture, be disastrous in a full load demand situation.

7.0 CONCLUSIONS

After five comprehensive chapters that delineate the assorted consequences on the overall performance swings due to considerations associated with coupling engineering workstations and mainframes, it should be obvious that it would be impossible to try and summarize to a single bottom line. It is clear that the residency question of the software is a multi-faceted one that requires pivotal understanding by the system configuration managers commanding their utmost understanding and efforts.

The worst possible situation that could exist is to have each individual engineering workstation organization develop its own operational software solutions in a system abyss. Interestingly, this is true even if there is zero interactive coupling of the operations that are distributed. Reiterated here are some of the reasons for this: the inability to use or reuse the developed code between disciplines; the various levels of software engineering proficiency which causes significant additional cost because of inept software designs and documentation; and the lack of a single point of control associated with managing the common network resource for input.

The best possible situation is where a software engineering organization is singularly responsible and knowledgeable of the effects thus described as well as all of the other critical considerations outlined in the preceding chapters. Then the residence of each new entry into the system software vault can be considered totally against the overall operational effectiveness avoiding the biased and limited examinations of a single discipline. Of course, this is the ideal situation, and quite often an organization must compromise the ideal because of what it usually calls an operational expediency. In any case, the preceding chapters can be summarized to some extent to provide the goal of a criteria question set.

7.1 Summary

As a summary, this chapter will furnish the system manager a question checklist set of criteria elements that were extracted from the earlier chapters. These are meant to be the stimulus for the enigmatic consideration that is necessary to conclude the software residency question. No single recipe exists that takes the input data and answers the question of residency. However, the questions are arranged so that if a "yes" answer results, then increased weight to the host as the residence is indicated. If a "no" answer is the result, then the engineering workstation weighing is increased. As was indicated earlier, however, there are some thresholds that can not be exceeded. In these cases, a firmer position can be taken. Questions marked with a single asterisk indicate that failure on these threshold items require no less than host computing power level. Items marked with a double asterisk indicate that the host can not meet the requirements in its aggregated operational status requiring off loading of an application onto the workstation. Other than these special question, the grey area on the residency question is generally considered quite large, and it can be

thought of as a non-linear problem solution that will yield better results after some modification and another iteration. For connection to the body of this report, each question is referenced to the appropriate chapter section number. This number can be used to locate the research discussion.

7.2 General Criteria Questions

As was stated, these general criteria questions will build a weighing factor that will hopefully indicate a trend for either the host or the workstation for the residency of the software. The questions that are not asterisked should be considered equal in influence on the decision. A reasonable, if not a somewhat arbitrary rule would say that if 70 percent of the questions are answered "no," then the workstation residency is the decision. If 70 percent are answered "yes," then the host residency is the most likely answer. Between these two limits, would be considered a range of optional consideration. Notice, that the influence in some areas is affected by restating the question with increasing levels of performance. This will either increase the number of yes or no answers and thereby strengthen the influence, or balance the yes and no answers and thereby nullify the influence.

7.2.1 System Considerations

There are a few system questions which need to be answered before the application specific questions can be considered. These questions are:

1. What type of operating system is being used on the host/mainframe? (5.1.1)

2. What type of operating system is being used on the workstation? (5.1.1)

3. Is there a load distribution algorithm for the system? (5.1.2)

4. If there is load distribution, is it load sharing? (5.1.2)

5. If there is load distribution, is it load balancing? (5.1.2)

6. Is there shared data on the host/mainframe? (5.3)

7. Is there shared data on the workstations? (5.3)

8. Is there a distributed file system? (5.3)

7.2.2 Computing Power

Chapter Three considerations are based on the computing power of the hardware. Before the questions can be answered, certain information needs to be furnished about the application, the workstation and the host. The information is all discussed in more detail throughout Chapter Three, however they are referenced here for convenience.

It is necessary to have the host and the workstation Sc values computed from the equation Sc = ((Wc*MFLOPS + Wo*MIPS)*MRC)$^{.5}$. This requires knowledge of advertised MFLOPS, and MIPS for both the workstation and the host. Knowledge from the application software yields the Wc, Wo, and the proper benchmarks to use to acquire the value of the MRC. Note that the benchmark nature needs to be as closely related to the actual application as possible and will probably be different in each application. The MIPS requirement of the software needs to be estimated as well as MFLOPS and the maximum time for execution. The MMC needs to be established as to what is required and available. The CPU ratings and the robustness factors for each computer needs to be calculated following the guidelines in sections 3.3.3 and 3.3.4 respectively. Following the collection of these items, then the next 24 questions can be addressed.

1. Is the host Sc 64 times greater than the workstation Sc? (3.3.1)

2. Is the host Sc 256 times greater than the workstation Sc? (3.3.1)

3. Is the host Sc 1024 times greater than the workstation Sc? (3.3.1)

4. Is the workstation value of MIPS available exceeded by the application MIPS requirement? (3.3.2)*

5. Is the aggregated application required value of MIPS exceeded by the host MIPS value? (3.3.2)**

6. Is the workstation value of MIPS exceeded by the aggregated application requirements? (3.3.2)*

7. Is the aggregated application requirements of MIPS with the addition of this application exceeded by the MIPS value of the host? (3.3.2)**

8. Is the workstation value of MFLOPS available exceeded by the application requirement? (3.3.2)*

9. Is the application required value of MFLOPS exceeded by the MFLOPS available on the host? (3.3.2)**

10. Is the workstation MFLOPS value available exceeded by the aggregated requirements with the addition of this application? (3.3.2)*

11. Is the aggregated requirement of the application program MFLOPS exceeded by the host value of MFLOPS available? (3.3.2)**

12. Is the maximum acceptable response time for the software less than 10 msec? (3.2.1)

13. Is the maximum acceptable response time for the software less than 100 msec? (3.2.1)

14. Is the maximum acceptable response time for the software less than 1.0 sec? (3.2.1)

15. Is the maximum acceptable response time for the software less than 10.0 sec? (3.2.1)

16. Is the required CPU rating estimated to be higher than the estimated workstation CPU rating? (3.3.3)*

17. Is the host CPU rating estimated to be higher than the estimated application required CPU rating? (3.3.3)**

18. Is the required MMC greater than 50 percent of the workstation MMC that is available? (3.2.3.1.1)

19. Is the required MMC greater than the workstation MMC that is available? (3.2.3.1.1)*

20. Is the available host MMC greater than the required application MMC? (3.2.3.1.1)**

21. Is the ratio of estimated number of computations compared to the number of other instruction executions greater than 100? (3.3.1)

22. Is the ratio of estimated number of computations compared to the number of other instruction executions greater than 100,000? (3.3.1)

23. Is the robustness factor of the host greater than .8? (3.3.4)

24. Is the robustness factor of the workstation less than .8? (3.3.4)

7.2.3 Software Development Issues

1. Is the application primarily a noninteractive function? (4.2)

2. If there is user interaction, is it only front-end queries? (4.2)

3. If there are queries, can they be separated from the calculations and passed as parameters to the calculation process? (4.2)

4. Do many disciplines require this application's function? (4.2)

5. Is the application only needed under special circumstances? (4.3)

6. Is the application a development tool? (4.3)

7. Is the response time requirement critical? (4.3)

8. Are there host applications which will need this application's output? (4.4)

9. Is there one other discipline which will use the application's output? (4.4)

10. Are there five disciplines which will use the application's output? (4.4)

11. Do all of the disciplines use the application's output? (4.4)

7.2.4 Control Considerations

1. Can the application run in the background? (5.1.2)

2. Does the application have communication needs with other applications on the host/mainframe? (5.2)

3. Is the application independent of applications on the workstation? (5.2)

4. Is the application dependent on applications on the host? (5.2)

5. If there are communication needs, does the application have infrequent communication needs with applications on the host/mainframe? (5.2)

6. If there are communication needs, does the application have periodic communication needs with applications on the host/mainframe? (5.2)

7. If there are communication needs, does the application, have heavy communication needs with applications on the host/mainframe? (5.2)

8. Does the application have infrequent communication needs with more than one workstation? (5.2)

9. Does the application have periodic communication needs with more than one workstation? (5.2)

10. Does the application have heavy communication needs with more than one workstation? (5.2)

11. Is the execution time a critical factor? (5.2)

12. Does the application require shared data access? (5.3)

13. Does the application access the shared data frequently? (5.3)

14. Does the application modify shared data? (5.3)

7.2.5 Networking Delay Impact

1. Are there more than eight message types required to be sent or received within this application? (6.4.2)

2. Are there more than 32 message types required to be sent or received within this application? (6.4.2)

3. Are the average message lengths greater than 100 bytes? (6.4.2)

4. Are the average message lengths greater than 1000 bytes? (6.4.2)

5. Are more than 10 messages per minute expected to be sent for this application? (6.4.2)

6. Are more than 1000 messages per minute expected to be sent for this application? (6.4.2)

7. Is the hardware network transmission speed less than 1 Megabit per second? (6.2)

8. Is the hardware network transmission speed less than 10 Megabits per second? (6.2)

9. Is the full protocol used to communicate at each level of the OSI model? (6.3)

10. Is the BER of the network estimated to be worse than one bit in 100,000? (6.4.1)

11. Is the BER of the network estimated to be worse than one bit in ten million? (6.4.1)

12. Does this application program have data messages for more than eight user destinations? (6.4.2)

13. Does this application program have data messages for more than 32 user destinations? (6.4.2)

14. Does this application have more than eight users who can make data inquiries? (6.4.2)

15. Does this application have more than 32 users who can make data inquiries? (6.4.2)

16. Is the network a connectionless service? (6.4.2.2)

17. Is the network utilization rate expected to be above 50 percent after adding this application? (6.4.2.2)

18. Is the greatest peak communication demand of this program above 10 percent of the network transmission speed? (6.4.2.2)

7.3 Methods For Applying The Criteria

When new applications are being created for a system which is already in operation, the principles discussed in Chapter Four concerning the

understandability and maintainability of software should be followed. If these methods are used from the beginning of the software design, it can lead to more efficient execution and resource utilization in the distributed system. A software developer must be conscious of decomposing the software into processes which perform different functions (i.e., separate user interface functions from calculation functions). The developer should also design the applications into modules which perform specific tasks and allow modules which perform common tasks to be reused by other applications. The ideas of task partitioning and task allocation discussed in Chapter Two should also be used in this case to better decompose the software into functions and modules which can then be partitioned to best utilize the system resources. Configuration management is especially important in a distributed system to control the duplication of code, task, and functions and to ensure that only validated software is executed during normal operation. Once a new application has been designed and developed using the principles above, answering the residency questions should be a fairly straight forward task.

In a system where the distributed system developed out of a single, time-sharing computer, answering the residency questions will be more complicated. The main reason for this being true is that the original applications were designed and developed for the time-sharing environment and not a cooperative one. The best approach to this problem is to analyze the current system and then apply the questions to each application which is being considered for migration. If the application can be easily modified to better conform to either the workstation or the host then these modifications should be contemplated and performed providing the modification cost is not greater than the expected benefit.

The residency decision for an universal application can be very simple or very complicated. In the simple cases, low usage requirements, developmental applications, non-real-time requirements, and local data requirements sway the decision heavily to either the workstation or the mainframe. In the complicated cases, the response time, data requirements, and frequency factors need to be evaluated collectively so that the most effective solution can be determined. Application of the residency questions for these types of applications will help to clear the picture in the complicated cases and verify the residency for the simpler cases.

Because of the tremendous swing in software application differences and hardware system construction let alone the great variety of design philosophies and the multitude of organizational possibilities, the residency direction that results from using these question guidelines should never be interpreted as an absolute measure. All of the questions are valid if not occasionally subjective and are always open to changes in the state of art over time. Still, using the question exercise above information will be gained that is critical to system performance, and the trend manifested will usually be reasonable. The exercise is significantly better than only a brief cursory review of the system. Further, after each iteration and application consideration the system manager will become more cognizant of the system limitations and

capabilities when considering the question of residency in a mixed host and engineering workstation environment.

APPENDIX A: ACRONYMS & DEFINITIONS

AT	Access time
AET	Accumulative execution time
ALU	Arithmetic and Logic Unit
BER	Bit error rate
BET	Benchmark execution time
CCITT	International Telegraph and Telephone Consultative Committee
CPU	Central Processing Unit
CHC	Channel capacity
CC	Clock cycle
CF	Clock frequency
CISC	Complex instruction set computer
DNP	Dedicated numerics processor
FPR	Floating point register
GPAMP	General purpose attached math processor
GPR	General purpose register
H/I	Hardware interruption
I/O	Input/Output
I/O/I	Input/Output interruption
IMC	Intermodule communication
IA	Interruption action
ISO	International Organization for Standardization
IPC	Interprocessor communication
LB	Load balancing
LS	Load sharing
LAN	Local Area Network
MLI	Machine language instruction
MMC	Main memory capacity
MRC	Maximum rate of computation
MCT	Memory cycle time
MAN	Metropolitan area network
MEPS	Millions of executions per second
MFLOPS	Million floating-point operations per second
MIMD	Multiple-instruction stream, multiple-data stream
MP	Multiple processors
NCH	Number of channels
OSI	Open System Interconnection
OS/I	Operating system interruption
PR	Precedence relationship
P/I	Program interruption
RISC	Reduced instruction set computer
SIMD	Single-instruction stream, multiple-data stream
SAMU	Smallest addressable memory unit
Sc	Speed of the computer
SMP	Special math processor
SMC	Standard math coprocessor
VA	Virtual address
VS	Virtual Storage
WAN	Wide area network

APPENDIX B: REFERENCES

Bersoff, E.H., Henderson, V.D., Siegel, S.G., Software Configuration Management, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1980, pp. 57.

Chafin, R., Martin, T. "DSN Human Factors Project Final Report", Los Angeles, California: University of Southern California, 1980.

Cheriton, D.R., "The V Distributed System," Communications of the ACM, Vol. 31, No. 3, March 1988, pp. 314-333.

Chu, W.W., Lan, M.T., "Task Allocation and PRecedence Relations for Distributed Real-Time Systems", Tutorial: Distributed-Software Engineering. Washington, D.C.: IEEE Computer Society Press, 1989, pp. 97-108 (Reprinted from IEEE Transactions on Computers, June 1987, pp. 667-679).

Chu, W.W., Hellerstein, J, Lan, M.T., An, H.M., Leung, K.K., "Estimation of Intermodule Communication (IMC) and Its Applications in Distributed Processing Systems", IEEE Transaction on Computing, Vol. C-33, August 1984, pp. 691-699.

Dahl, O., Dijkstra, E., Hoare, C., Structured Programming. New York: Academic Press, 1972, pp. 1-82.

Dias, D.M., Iyer, B.R., Robinson, J.T.,Yu, P.S., "Integrated Concurrency-Coherency Controls for Multisystem Data Sharing", IEEE Transactions on Software Engineering, Vol. 15, No. 4, April 1989, pp. 437-448.

Dias, D.M., Iyer, B.R., Yu, P.S., "Tradeoffs Between Coupling Small and Large Processors for Transaction Processsing," IEEE Transactions on Computers, Vol. 37, No. 3, March 1988, pp. 310-320.

Dongarra, J.J., "Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment," Argonne National Laboratory, Argonne, Illinois, pp. 47-69.

Dongarra, J.J., Martin, J.L., Worlton, J., "Computer Benchmarking: Paths and Pitfalls," IEEE Spectrum, July 1987, pp. 38-43.

Efe, K, "Heuristic Models of Task Assignment Scheduling in Distributed Systems", Tutorial: Distributed-Software Engineering. Washington, D.C.: IEEE Computer Society Press, 1989, pp. 89-95 (Reprinted from Computer, June 1982, pp. 50-56).

Ein-Dor, P., "Grosch's Law Re-Revisited: CPU Power and the Cost of Computation," Communication of the ACM, Vol. 28, No. 2, February 1985, pp. 142-151.

Falk, H., "Development systems evolve toward integrated, host-independent solutions," Computer Design, April 1, 1988, pp. 44-50.

Faulk, S.R., D.L. Parnas, "On Synchronization in Hard-Real-Time Systems", Communications of the ACM, Vol. 31, No. 3, March 1988, pp. 274-287.

Galitz, W.O., Handbook of Screen Format Design. Wellesley Hills, Massachusetts: QED Information Sciences, Inc., 1985, pp. 16-20.

Gilfoil, D.M., "Warming Up to Computers: A Study of Cognitive and Affective Interactions Over Time," Proceedings: Human Factors in Computer Systems, pp. 245-250. Gaithersburg, Maryland, March 15-17, 1982.

Huang, J.P., "Modeling of Software Partition for Distributed Real-Time Applications", IEEE Transactions on Software Engineering, Vol. 11, No. 10, October 1985, pp. 1113-1126.

IBM Corp., IBM 3081 Functional Characteristics, IBM Pub. No. Ga22-7076-7, Seventh Edition, 1986.

Kraemer, K.L., King J.L., "Computer-based Systems for Cooperative Work and Group Decision Making," ACM Computing Surveys, Vol. 20, No. 2, June 1988, pp. 115-146.

Krueger, P, M. Livny, "A Comparison of Preemptive and Non-Preemptive Load Distributing", Proceedings Of The Eighth International Conference on Distributed Computing Systems, Washington, D.C.: IEEE Computer Society Press, 1988, pp. 123-130.

Kusmanoff, A.L., "Real Time Bearing Estimation in a Multi-source Environment Using Multi-processor, Multi-algorithmic Acceleration," Ph.D. Dissertation, Oklahoma State University, May 1989.

Litzkow, M.J., M. Livny, M.W. Mutka, "Condor - A Hunter of Idle Workstations", Proceedings Of The Eighth International Conference on Distributed Computing Systems, Washington, D.C.: IEEE Computer Society Press, 1988, pp. 104-111.

Martin, J., C. McClure, Structured Techniques For Computing. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1985, pp. 16.

Mok, A.K., "The Decomposition of Real-time System Requirements into Process Models," Proceedings of the Real-time Systems Symposium, IEEE Computer Society, Washington, D.C., 1984, pp. 125-134.

Perry, T.S., Zorpette, G., "Supercomputer Experts Predict Expansive Growth," IEEE Spectrum, February 1989, pp. 26-33.

Schneidewind, N.F., "Distributed System Software Design Paradigm With Application to Computer Networks", IEEE Transactions on Software Engineering", Vol. 14, No. 4, April 1989, pp. 402-412.

Shatz, S.M., J. Wang, Tutorial: Distributed-Software Engineering. Washington, D.C.: IEEE Computer Society Press, 1989, pp. 58-59.

Smith, J.E., "Characterizing Computer Performance With a Single Number," Communication of the ACM, Vol 31, No.10, October 1988, pp. 1202-1206.

Stankovic, J.A., Reliable Distributed System Software, Silver Spring, MD: IEEE Computer Society Press, 1985, pp. 83-84, 135-138.

Rauch, K., "Math Chips: How They Work," IEEE Spectrum, July 1987, pp.25-30.

Thierauf, R.J., Effective Management Information Systems. Columbus, Ohio: Bell & Howell Company, 1984, pp. 218-222.

Vick, C.R., C.V. Ramamoorthy, Handbook of Software Engineering. New York, New York: Van Norstrand Reinhold Company, 1984, pp. 656-674.

Wirth, N., "Program Development by Stepwise Refinement," Communication of the ACM, Vol. 14, No.4, April 1971, pp. 221-227.

THE ROLE OF THE HOST IN A COOPERATING MAINFRAME AND WORKSTATION ENVIRONMENT

Volume II

NASA Grant Number NAG 9-341
SwRI Project Number 05-2769

Submitted to:
NASA-Johnson Space Center
Houston, Texas

Prepared by:
Antone Kusmanoff, Ph.D.
Nancy L. Martin

Southwest Research Institute
6220 Culebra Rd., P.O. Box 28510
San Antonio, TX 78228-0510

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

1.0 INTRODUCTION

1.1 Purpose

Since the beginning of manned space flight, the Mission Control Center environment at NASA/Johnson Space Center (JSC) has been dependent on a mainframe concept to do all of the computation and display of information necessary for the support of a space mission. As discussed in the first volume of this document, advancements made in computer systems have prompted a move from centralized computing based on timesharing a mainframe computer to distributed computing based on a connected set of engineering workstations. The MCC has not been immune to this move. As systems were upgraded to bring new technology into the MCC, the overall system at NASA/JSC has evolved.

The shift to distributed computing from centralized computing has led to challenges associated with the residency of application programs on the system. In this volume the residency guidelines established in the first volume will be applied to the MCC environment and recommendations concerning the residency of applications in a cooperating host/mainframe and workstation environment will be presented. Currently, there is a tremendous amount of software executing on the Real Time Host (RTH) and the Flight Support Host (FSH) in the MCC. As workstations are introduced into the MCC environment, consideration must be given to the distribution of current host software between the workstations and the host. The purpose of this research is to develop an understanding of how the role of a host and workstation may evolve in the future due to advancements made in workstations and distributed systems. The ultimate goal of this research is to provide guidelines to aid in the development of a software environment which allows the optimum usage of all technologies currently available, as well as providing a means to integrate new technologies as they are developed.

1.2 Document Organization

The recommendations for the residency of applications in the MCC will be presented in the following chapters. This document has been arranged into this introductory chapter, a chapter on the current hardware environment of the MCC, a chapter concerning the migration of current applications and the allocation of global functions, and a chapter discussing development and placement of new applications. A concluding chapter will offer recommendations for the direction of the MCC in the future.

The hardware environment chapter will concern itself with identifying the hardware portion of computing power of the current architectures available in the MCC. This portion will be addressed using the issues and categories discussed in Chapter Three of Volume I. The combined mainframe/workstation performance is a conclusion of the integration of the hardware, its organization, and architectures interacting dynamically with the software operating system, and applications programs.

The third chapter will begin the discussion of the residency question for applications which are currently executing on the hosts and workstations. This chapter will take categories of functions and, using the questions from Volume I, objectively recommend the location of those applications based on the questions for which answers are available and intuitive reasoning. This chapter also discusses the categories of universal applications such as configuration management.

In Chapter Four methods for developing and assigning new applications will be discussed. This chapter will take into consideration the necessary approaches in software development which will help resolve the residency question.

In the final chapter, recommendations will be made for the future direction of the architectures in the MCC. These recommendations will be based on types of functions which need to be carried out for the MCC to remain fully operational and still maintain an efficient usage of the available hardware.

2.0 HARDWARE ENVIRONMENT

In Volume I the computing power was defined as an integrated combination of the hardware performance factors and the software correlation to the hardware factors. This chapter will provide the computer hardware decision criteria to contrast between the NASA/JSC mainframe and the engineering workstations when considering the residency of the software application.

The first step in assigning software applications in a specific environment of cooperating mainframes and workstations is to derive the estimate of the individual computing powers for each different computer in the network. Referring to the last section of the third chapter of Volume I, it is seen that the computing power is related to a large number of hardware metrics which are grouped into four segments, the speed factor, the capacity factor, the CPU rating factor, and the robustness factor. Hence, in the case of the NASA/JSC MCC, with the mainframe being the IBM 3081 and the workstations being MASSCOMP 6650s, the hardware configuration metrics of Chapter Three of Volume I will be obtained from the manufacturer's literature and other published data on these computers. With this done, it is possible to estimate the relative computing powers available at each level. Following that, the set of residency questions from Chapter Seven of Volume I can be approached appraising each specific software application.

Within the MCC complex there are five mainframe computers, IBM 3081s, which are available to act as the Real-time Host (RTH) during an actual shuttle flight. A typical mission has one mainframe functioning as the Mission Operations Computer (MOC) and another as the Dynamic Standby Computer (DSC) which runs in parallel with the MOC. This parallel processing is included to increase the reliability of the system. The DSC is running synchronously, instruction by instruction, with the MOC and is capable of taking over as the active MOC should a failure occur. During a flight, there is also another mainframe used simultaneously as the Flight Support Host (FSH). The computational activity occurring on the FSH is relatively independent of the processing on the MOC/DSC combination. In that configuration, the "host" can be considered a multiple mainframe solution. This multiprocessing influence is not taken directly into consideration in this chapter, just as the total number of workstations is not an included consideration. This comparison is single mainframe versus a single workstation. Also, the IBM 3081 has a dyadic processor that will be dealt with as a single processor system. This is because the main function achieved with the dual processors is increased reliability.

The engineering workstations which are being integrated into the JSC MCC are MASSCOMP 6650s. The 6650 is considered a true multiprocessor machine as long as the applications and the operating system are able to take advantage of the hardware features. This installation is supporting two processors, and work is being completed to take advantage of the multi-

processing capability at the operating system level, hence the workstations will be considered multiprocessor capable.

The four factors mentioned above, and all of the elements in each factor, will be covered in the following sections. Each entry will have a discussion and a breakout of the metrics for each of the computers.

2.1 Speed Factors

The first speed factor is the performance obtained based on the computers executing benchmark programs. In Volume I it is was stated that it is possible to include this component measure when the benchmark program is known to be reasonably close to the application need. It was stated to be even better when an appropriate benchmark is available that was specifically developed along the same nature as the user application requirements. However, in this case there are a multitude of software applications that need to be considered. Hence, a set of benchmark programs with a mix of operational types would be recommended to be used to obtain a Maximum Rate of Computation (MRC) value for each system. It would be reasonable to select the benchmark set such as the one provided by the National Institute of Standards and Technology (formerly the National Bureau of Standards). The programs, known as the NBS benchmarking collection, can be accessed by way of Arpanet. Doing this, some sort of averaging or weighted joint measure would be necessary to arrive at a single MRC using all of the required benchmark programs. Of course, it is still possible that the actual operational system performance will be incorrectly estimated. Since the benchmark data is only one element of the speed factor, and the speed factor only one part of the overall computing power estimate, a small variation between the model and reality is not unreasonable. In any case, currently no reasonable MRC data is presently available that meets the requirements stated above. Because of this, the assignment for both MRC values is N/A, standing for not available. It would be inappropriate to estimate this parameter from other data provided directly from the manufacturer such as the MIPS (Millions of Instructions Per Second) and FLOPS (Floating Point Operations Per Second) ratings. Further, these values are already included in the overall computing power estimate equation for speed of the computer, Sc. The primary disadvantage of not having the MRC is that the Sc value used the MRC as a damping factor in the geometric average with the other speed data. In this case it will be necessary to only include the weighing factors and the MIPS and FLOPS ratings that follow.

The next speed factors are the published values of MIPS and MFLOPS. Also included below, but not directly entered into the speed factor is the Clock Frequency (CF) of each separate machine. All of this information is available from the manufacturers and other sources such as the Argonne National Laboratory. The MIPS and CF values were taken directly from NASA and manufacturer's specifications. The value for MFLOPS was extrapolated from data published in 1988 by the Argonne National Laboratory. The performance was measured on the MASSCOMP 5600 with the floating point accelerator and was reported to be .33 MFLOPS. Using the advertised improvement of the 6650 over the 5600, 7 million Whetstones/second versus

3 million Whets, a conversion factor of 2.333 was derived. Using this factor, a prediction of .77 MFLOPS results for the MASSCOMP 6650. This comparison data is based on solving a system of linear equations with LINPACK in full precision using Fortran.

	IBM 3081	MASSCOMP 6650
MRC=	N/A	N/A
MIPS=	15.6	13.3
MFLOPS=	2.1	.77
CF=	41.67 MHz	33.0 MHz

It turns out that the 1980 designed mainframe and the 1989 designed workstation have a speed performance in the same general neighborhood. Of course there are three more factors to follow before the comparison of computing powers can be made. Also there was no correspondence possible based on an equitable (application oriented) benchmark system, which could highlight a possible speed difference. In any case, these values do conclude the metrics associated with the speed factor. Using this information and the weighing factors that arise from the software analysis, the speed factor, Sc, for each computer can be approximated to be used in the set of questions.

2.2 Capacity Factor

The quantities of MRC, MIPS, or MFLOPS are speed related, but they also have a thresholding nature. That is, these hardware factors are directly related to hard limitation thresholds associated with software demands. If the capacities associated with an application cannot be met, the computing power of the system falls short of the software requirements. Of course when considering these measures as thresholds, it is necessary to have the application program requirements explicitly established. The capacities associated with engineering workstations can be augmented by the system requirements when software distribution requires an interaction with application software at other locations.

Another capacity constraint, the Main Memory Capacity (MMC), may be a cost factor, a hardware compatibility factor, or both. In any case, it is a parameter that is provided by the computer hardware and demanded upon by the operational software. Exceeding the crossover of the demand and constraint is incompatible with successful operation and can be considered a hard limit threshold. Large storage capacities using virtual memory, by applying mapping overlays are used to extend MMC, however the installed values are as follows:

	IBM 3081	MASSCOMP 6650
MMC=	16 MB	32 MB

2.3 CPU Rating Factor

The objective thresholding quantities just addressed are very different from the softer or objective rating given to the CPU. As was stated when it was developed, the CPU rating is to be used in conjunction with the

earlier established measures of power to help differentiate between different CPU architectures. The six elements assigned to build the composite CPU rating will each be rated below.

	MAX	IBM 3081	MASSCOMP 6650
1. INSTRUCTION SET			
CISC	10	10	10
RISC	20		
2. INTERRUPTION ACTION			
BASIC	2	2	2
P/I	2	2	2
I/O/I	2	2	2
H/I	2	2	2
O/S/I	2	2	2
3. REGISTER CAPABILITIES			
GPR 1-5	3		3
GPR 6-10	5		
GPR 10 OR ABOVE	7	7	
FPR ANY ADD	3	3	
4. SPECIAL MATH PROCESSORS			
SMC	10		
DNP	15		
GPAMP	20	20	20
5. MULTIPROCESSORS			
$2*LOG_2(P)$	20		2
6. I/O CAPABILITY			
NCH 1-10	5		5
NCH 11 OR ABOVE	10	10	
CHC 1-10 MBPS	5		5
CHC 10 MBPS UP	10	10	
CPU RATINGS		70	55

2.4 Robustness Factor

In Volume I several hardware factors were related to what was called the robustness of the computing power. These factors are applied as weights correlated to the difficulty of use, sensitivity to software environment

changes, and maturity of the technology. These are also soft measures with some subjectivity included in the value. Remember that some of the factors that are included were inputs to other components of the computing power. The closer the robustness factor is to one, the higher the probability of the computer system reaching the level of power anticipated or needed. The final robustness factor is the product of the assigned weights. If the computer system does not have the optional factor, it is considered to be one (1) for that system. The application of the robustness factor against the residency questions can be accomplished after its value is computed as follows:

	MAX	IBM 3081	MASSCOMP 6650
1. CACHE MEMORY	.99	.99	.99
2. SAMU			
BYTE	.9999	.9999	
MULTI-BYTE	.999		
WORD	.99		.99
3. VS	.99	.99	.99
4. SMP			
SMC	.99		
GPAMP	.99	.99	.99
DNP	.9		
5. INSTRUCTION SET			
RISC	.9		
CISC	.9999	.9999	.9999
6. MP	.9		.9
ROBUSTNESS PRODUCT		.970	.865

2.5 Summary

The information above reports that considering the systems in the MCCU 1) the new workstation is approaching the speed of the older mainframe, 2) the MMC of the workstation is greater, 3) the CPU of the mainframe is higher rated, and 4) the performance predicted by the mainframe has a higher probability of occurring than the performance of the workstation.

The above data can be used to answer some of the questions at the end of Chapter Seven of Volume One. This chapter furnishes the system manager the set of criteria elements that were extracted from the hardware configuration. This information is summarized in a System Information Summary included in Appendix B. There is not always a simple answer to the residency question, but this information was meant to be the stimulus for the complex considerations that need to be made. Volume I clearly stated that a single recipe does not exist which takes the input data and

answers the question of residency. Certainly, these hardware factors are only a fraction of the inputs. Of course, there are some thresholds that cannot be exceeded. In these cases, a stronger position was taken in the question area. Other than these special situations, the undecided area on the residency question is quite large and is a subjective issue. However, the information that has been gained in this analysis is important to system performance, and the trend manifested is reasonable. The limitations and capabilities that come out of the hardware considerations are extremely valuable when considering the question of residency in a host and engineering workstation environment.

3.0 CURRENT ENVIRONMENT MIGRATION

In the MCC, the cooperating mainframe and workstation system developed out of a time-sharing environment which makes answering the residency question more complicated. The difficulty in this situation is that the original applications were designed and developed for the time-sharing environment, not a distributed, cooperative one. A practical approach to this problem is to analyze the current system and then apply the residency questions from Volume I to each application/subsystem which is being considered for migration.

Currently, there is a tremendous amount of software executing on the Real-Time Host (RTH) and the Flight Support Host (FSH) of the MCC. One of the main objectives of this research has been to identify the applications which execute on the hosts, determine which disciplines use them in some capacity, apply the residency questions as completely as possible with the available information and then recommend whether or not a host application should be moved to the workstations for execution. On the other hand, there has also been a large amount of software written for the workstations. In most cases, this software is probably best suited for the workstation because the applications are discipline-specific. There may, however, be a few cases of workstation applications which may be better suited to run on the host mainframes.

The residency questions from Volume I have been applied to a number of applications and subsystems which are currently executing in the MCC. As many residency questions as possible have been answered in Appendix B of this document. Using the knowledge that could be gleaned from this exercise, recommendations have been made concerning where some of the applications/subsystems should reside. Further investigation and more detailed answers will have to be collected before more of the residency questions can be answered.

3.1 Host Applications/Subsystems

Even with the most current advancements, today's workstations do not have the processing power to perform all of the functions which are currently executed on the RTH and FSH. Even if they did, to rewrite all of the mainframe software in workstation code would be a timely and costly venture. The mainframes are well suited for certain tasks and should continue to be employed for real-time data reduction and for maintaining a massive database of statistics from many flights (current and historical). Both the raw real-time data from the spacecraft and the processing results from the hosts will be available on the Local Area Network (LAN) for the workstations to use. In this manner, as workstations become more powerful they may take on more of the host's functions or they may use their added power for other tasks. The growth path is quite flexible and those decisions can be made as the workstations advance and new functions are conceptualized.

The RTH and FSH of the MCC are IBM 3081 machines. During an operational flight, there are two mainframes up and running the RTH applications. One mainframe acts as the Mission Operations Computer (MOC) and one is the Dynamic Standby Computer (DSC). The DSC runs in parallel with the MOC and provides a reliable backup to the MOC. The two mainframes share their data on a Direct Access Storage Device (DASD). The role of the FSH is to provide the Near Real-time Telemetry (NRT) Data Reduction which serves as the telemetry data retrieval system for the MOC and Payload Operations Control Center (POCC). The other applications which execute on the FSH are analysis and monitor type functions.

3.1.1 Telemetry Functions

The primary function of the MOC/DSC computers is the commutation and calibration of raw telemetry data from the Network Data Driver (NDD). The MOC receives the raw data from the space vehicle and then performs the calculations necessary to provide the values to the disciplines which have requested the data for display to the Flight Controllers.

The Telemetry application unpacks data sent from the NDD, converts that data to engineering units, performs calibration and computations, and stores the results of these processes in the Intermediate Data Array (IDA) which is then output on the RT-LAN. Initially, processed telemetry is retrieved by Processed Parameter List (PPL) requests from the workstations. The RTH processes cyclic telemetry data on a one second basis. The RTH will also process application/user requests for telemetry data.

The NDD downlinks raw telemetry data messages at an approximate once-per-second rate. The host telemetry is required to build and output multi-cast (via the RTLAN), a MOC Telemetry Message (MTM) for every NDD data stream that it is processing. The calibrated data must be made available to the workstations 5 seconds after its receipt from the NDD.

Because of the need to store this data for historical purposes this function should remain on the host. Although applications on the workstations are capable of processing raw data themselves, not all do. A number of them prefer using the preprocessed shared data provided by the host.

3.1.2 Near Real-time Telemetry (NRT) Retention

Near Real-time Telemetry (NRT) data is stored continuously on the RTH by the NRT Retention application. This application stores unprocessed full rate Space Transportation System (STS) data to the NRT data base and archival tape. Retention will support data on a flight basis from the Telemetry Preprocessing Computer (TPC).

The data routed to the MOC may be retained in the database. A block of data will be shipped to Retention from the MOC via Multiple Virtual Storage (MVS) Cross Memory services. As a result, the Retention job must reside in the same host as the MOC. When supporting a TPC configuration,

a second Retention job will run in parallel on the DSC. The DSC Retention job will retain to tape but not to disk. Upon notification from MOC Telemetry, the roles of the MOC Retention and DSC Retention jobs will switch.

The size of the data base will depend upon the amount of DASD available. The worst case loading requires support of 3 STS streams for one flight. A stream is comprised of up to 16 K bytes data plus some header and status bits. Data should be available for retrieval no later than 5 minutes after it is received at the host. This application is projected to need .18 MIPS for execution.

Since this application requires a significant amount of storage capacity and a strong reliability factor there is no reason to move this application to the workstation. If this application were moved to the workstation level then each workstation would need a copy of its data base or some method of maintaining consistency within the data base of NRT data.

3.1.3 Near Real-time Telemetry (NRT) Reduction

The Near Real-time Telemetry (NRT) Reduction subsystem runs on the FSH and retrieves a block of retained telemetry data from the NRT data base upon request from a workstation. Data values are retrieved using Measurement Simulus Identification (MSID) values. Parameter list data values retrieved from the data base may be transmitted to the user at a workstation. In addition, report generation of NRT data can be generated. These parameter list and report generation retrievals are referred to as reduction runs.

Since the telemetry data is output to the data base in raw form, upon retrieval the required data may optionally be unpacked from its message, calibrated, limit sensed and have the necessary computations performed before it is shipped to the requesting workstation. This application is projected to require 2.5 MIPS for its execution.

This application should not be moved off of the host because of the universal need for its output and its access of the large NRT data base.

3.1.4 Trajectory

The Trajectory application provides trajectory related information processing in support of the flight controllers. Trajectory processing functions are performed on demand as a result of a flight controller's request, and automatically in response to incoming network radar data and cyclic time queues. Some of the applications are just data management, but there are a large number of calculation operations.

Trajectory data is accessible to any workstation that is connected to the LAN through the MITS-approved Data Control Lists (DCLs) using the Trajectory Data Retrieval (TDR). This application is projected to require 1.2 MIPS during its execution.

This application should not move, because of a common demand for its output and because of the reliability factor of the host.

3.1.5 High Speed Guidance, Navigation and Control (GNC)

There is a clear line between high speed and low speed applications for navigation. Two examples of trajectory functions which fall in this category are the Satellite Acquisition Table (SAT) and the Landing Opportunities application. The SAT calculates when the shuttle is going to pass a certain point. The Landing Opportunities application computes landing opportunities if the need arises.

The high speed trajectory functions are performed during ascent and re-entry and need to remain on the host in order to keep pace with rapid progression through the stages of these mission phases.

3.1.6 Low Speed Guidance, Navigation and Control (GNC)

The low speed GNC functions perform the same operations as the high speed functions. However, the low speed trajectory functions execute during the slower paced orbit phase. Because of this pace, the low speed functions could move to the workstation. In order to move to the workstation, however, there would still be a need for access to a common table of data (not large). If there is a reliable and efficient communication mechanism, one table could be maintained on the host and accessed by the workstation. This would require modification to the current code in order to provide access to the data from the workstation.

3.1.7 Vector Propagators

Two examples of vector propagators are the Analytic Ephemeris Generator (AEG) and ENCKE. These functions are used primarily by the Trajectory discipline to perform numerical integration, but are sometimes used by other disciplines.

They could probably be taken off of the host and their functionality downloaded to each discipline which uses it. If they used shared data on the host, then modifications would need to be made to access the necessary data.

3.1.8 Network Communications (NETCOM)

The NETCOM provides monitoring and/or control of the Network Communications Interface Commons (NCIC's), TDRSS, Dump Data Handling Subsystem (DDHS), remote sites and the Network Output Multiplexer (NOM). The NETCOM also provides metering of data output to the NOM. It submits configurations and reconfiguration and keeps status on telemetry, trajectory, and software check-out. It is a table driven application used by the Operations Support Team (OST). This application also monitors the Digital Television Equipment (DTE), Manual Entry Device (MED), and Push Button Indicator (PBI) hardware and their emulations.

NETCOM parameters are retrievable via the Generalized Data Retrieval (GDR) capability in the Command Control System/Control (CCS/C) with the following supported rate options: Cyclic - request a data value message for a PPL to be output to the requesting workstation at the specified cyclic rate; Single shot - request a data value message for a PPL to be output to the requesting workstation only one time; Data change - request a data value message for a PPL to be output to the requesting workstation when data for the PPL is first requested and whenever data values for parameters specified in the PPL change.

Since this application is only used by one discipline, the Operations Support Team (OST), and doesn't use any shared data, it would be possible to move this application to the workstation level provided there is sufficient access to the status and control data for the systems which it monitors. If there are other disciplines which require this application's results, a communication mechanism would be needed to send the results over the GPLAN when requested.

3.1.9 Command Control System/Control (CCS/C)

Data is received from the real-time hardware through the operating system into the CCS/C application. The CCS/C logs this data and routes it to the appropriate applications. Data is also received from these applications. The CCS/C logs and sends data through the operating system to the real-time hardware. There is an additional CCS/C function known as GDR. The GDR provides a data retrieval service for the workstations to request data from the host and monitors the data flow. The CCS/C Subsystem also provides the emulation of PBI's, DTE's, and MED's to the workstations. The emulation of MEDs allows input from the workstation which will result in processing as if the request was input from the dedicated CRTs which are currently used as the MEDs. Output advisory/error messages that result from MEDs that are entered from workstations will also be output to the workstations from which the MEDs were entered. The DTE emulation will allow requests for display data to be handled via CCS/C services to support display requests from the workstation environment. PBIs are emulated by an input from a workstation which results in processing as if the PBI was input from a console. Output advisory/error messages that result from a PBI entered from a workstation shall be output to the on-line printer and to the on-line monitor in the same manner as if the PBI was entered from a console. Advisory/error messages that are output to the on-line monitor shall also be output to the workstations from which the PBI was entered.

This subsystem should not be moved from the host since its purpose is to serve as an interface to the host from applications which execute on the hosts and the workstations.

3.1.10 Command Application

The Command application performs Shuttle ground command software functions in the MCC. These functions provide the capabilities to generate, format

access for viewing, and transmit vehicle commands in response to inputs from flight control workstations. The Command application also generates network management commands and validates the success of transmission based on vehicle telemetry downlink and network responses. The Command application also validates commands uplinked to the Orbiter.

There is a universal need for the command verification. It should remain centralized on a host in order to maintain control of the transmission of commands to the space vehicle.

3.1.11 Network Support Software (NSS)

The NSS resides in both the RTH and FSH as the interface between the host resident applications and the LAN. The primary function of the NSS is to interface the host computer or application with the RT-LAN or GP-LAN. NSS will log and/or trace (in an NSS trace table) all major events (such as error conditions, and some portion of each Communications Server (COMSERVER) input/output request). The trace table will also be logged to tape or disk. Both the log data and trace table data will contain time tag information. Host applications requiring NSS services have their own channel interface to the LAN COMSERVER. NSS supports host application-requested creation and destruction of virtual circuits. NSS maintains a list of valid node names that each host application may support and maintains a list of the valid subchannel addresses that NSS will support for each LAN COMSERVER. NSS provides the user the capability to print and/or view NSS displays on the host terminals and provides an interface to the computer operator for the input of commands. These commands may be in the form of key word commands or MED commands. Note that no responses (or displays) will be returned to the computer operator's console. Additionally, the computer operator's console will receive text messages from NSS describing certain cases of error and advisory conditions.

This application requires a maximum of .23 MIPS for its operation and 4.17 MB of memory. This application should not be moved since its purpose is to serve as the interface between host applications and the LAN.

3.1.12 Configuration Management (CM)

CM is primarily a library function of storing, downloading and uploading all software elements (data, source files, object files, computations, etc.) for mission, Independent Verification (IV), simulation (SIM), and development operations. Configuration Management provides the functions necessary to maintain and control the various operational software configurations for operational workstations. This includes the establishment of a centralized system library and control of access to the elements in this library. Interfaces between the workstation and host system users provide the method for establishing and maintaining the library elements.

CM processing is divided into two major parts: a Host (global) function and a Local (workstation) function. The Host portion maintains the

central library and distributes elements to requesting nodes. The Local portion controls the local environment and communicates with Host CM for upload/download of library elements.

The functional requirements for the Host CM processing consist of the database management, real-time host services, system structure management, and database report processing. To accomplish the required processing for these components, Host CM is required to provide both real-time support and nonreal-time support.

The real-time support consists of the Host Real-time Services and portions of the Database management functions. All processing is accomplished between the Host CM function and workstations via the Local Area Network (LAN) interface. This processing is generally referred to as the 'real-time' portion of Host CM.

The nonreal-time support consists of the System Structure Management and portions of the Database Management function. All processing will be accomplished between the Host CM function and a terminal via an interface method. This processing is generally referred to as the 'structure management' portion of Host CM.

Host CM controls the majority of the workstation based software elements. Host elements, front end components, communication server components, host operating system, etc. are not to be considered part of the CM library.

Host CM processing maintains a centralized library of the elements needed by workstation processing to provide operational system support. This central library will be the major part of the CM system and is managed via an interface with a Commercial Off-The-Shelf (COTS) Data Base Management System (DBMS). The CM database will contain the actual workstation elements as well as supporting information identifying usage of each element. Other parts of the CM database will provide the basis for control of access and use of the library elements.

Real-time Host CM processing will provide an interface between the operational workstations and the CM host. The primary function of this processing is to allow workstation software elements to be transmitted to the workstation for real-time configuration and to allow workstations to transmit new or modified elements for storage within the Host CM database. The real-time Host CM function will be available as a separate application from the other host resident jobs (MOC, DSC, NRT, etc.). A real-time Host CM job accessing a single Operational database is required to be active in only one host machine at any given time, due to the restrictions of available COTS DBMS products. However, more than one Host CM job will be allowed to access the Operational Database(s) at any one time, in order to support dual operational activities. (If this capability requires the use of dual Operational Databases, then procedures shall be implemented to keep the contents of both databases synchronized).

The FSH will be the primary target environment for Host CM job executions. Procedural controls and system availability may make it necessary to run the Host CM function in the Real-time Host (RTH) when the FSH is not available.

The structure management component of CM provides the capability to define one of three databases for access at session initialization. It also provides the capability to define and update valid user definitions, allow user definitions to identify predefined lists of elements to be downloaded as a result of the workstation sign on request, define and update valid group definitions, define and update valid function definitions, define and update valid access control information, update flight and certification level qualifiers for existing elements (without having to upload the element data from the workstation), and create and store reconfiguration products on the CM database. The structure processing will be performed independently from the real-time Host CM processing. These functions will be accomplished via a terminal interface to the structure management portion of the Host CM application. Inputs to Host CM will consist of processing requests related to creating and maintaining the CM database structures required to support the central element library. Outputs will consist of screen information showing the state of the requested structure.

This subsystem requires a maximum of 3.9 MIPS for its execution and 2.71 MB of memory. Since it maintains a library of all workstation software, CM should remain on a centralized system such as the host in the current MCC configuration.

3.1.13 Shuttle Configuration Analysis Program (SCAP)

The Shuttle Configuration Analysis Program (SCAP) is used by the flight control team in the assessment and resolution of in-flight anomalies. SCAP consists of a series of programs and databases containing configurations of electrical components, busses and orbiter sensors for the shuttle and its payloads. SCAP provides an automated, interactive system for maintaining and accessing the databases. As detected component failures are input to SCAP, other related failures and single-point failures are output, including the accumulation and interaction of multiple failures. The primary input to SCAP is anomaly information resulting from flight status monitoring activities. The primary output from SCAP are displays and/or reports detailing the failure effects or configuration information. Access to the SCAP system is currently provided to users via the MITS LAN and GP LAN to MASSCOMP workstations. Users performing SCAP processing via a MASSCOMP/C3 Integral Graphics Processor (IGP) are provided an automatic log-on capability, which will support up to four IGPs per workstation initially. The user will not be required to manually establish a Time-Share Option (TSO) session. SCAP will automatically come up when it is selected from the process pull-down menu under the Workstation Executive (WEX).

There are two programs included in the SCAP subsystem - the Failure Analysis Program (FAP) and Instrumentation. The FAP serves the flight

controller by assessing the multiple system/component failures, next worse failures, loss of functional capabilities, loss of redundancy, and single point failures based upon a given input failure case. It uses a logic database to show the relation of each shuttle component to various functional capabilities and systems. It uses data from the Shuttle Data Tape that describes the various shuttle components involved.

Instrumentation provides the capability to perform retrievals of SCAP channelization, SCAP Line Replaceable Unit (LRU), full telemetry, and calibration. Instrumentation enables the flight controller to selectively retrieve information of specific orbiter parameters. It uses a flight dependent instrumentation database created from the Shuttle Data Tape (SDT), the Payload Instrumentation Parts and Components List (P/L IPCL), the System Software (S/S) IPCL, and the Payload Data Tape (PDT).

The SCAP program is used by a few disciplines and runs on a huge database. It may be able to be moved to the few workstations which use it if the memory capacity of the workstation is sufficient to hold the database. This application requires a maximum of 4.6 MIPS for execution and has a targeted response time of 30 seconds.

3.1.14 Fault Message Application

The Fault Message Application is a table look-up application used by the Data Processing Support (DPS) discipline. Other disciplines request to see the output of it, but it is a trivial application for a mainframe. The DPS workstation could handle this application if there was a communication mechanism to share output between workstations.

3.1.15 Scratch Pad Line

This application decides where crew input to on-board computers will go. It is mostly table look-up and is used primarily by the DPS discipline. The workstation could handle this application, if there was a communication mechanism available to share the output when requested.

3.1.16 Orbiter Attitude

The Orbiter Attitude application is used by many disciplines. It calculates the attitude of the vehicle and stores the result in the host for access by disciplines which need it.

This application could be moved to the workstation if there were a communication mechanism set up to provide the output to those other disciplines which may want it.

3.1.17 Time Storage

The calculation of all the times used during a mission are done on the FSH and stored in the MOC for access by all applications and disciplines. These times include the Greenwich Mean Time (GMT) and Mission Elapsed Time (MET).

Since this application provides data necessary to every discipline and needs to be processed frequently, it is recommended that the function to compute the times remain on the host.

3.2 Workstation Applications

As the workstations have migrated into the MCC, there have been many applications written to run on them. Many of these applications are discipline-specific and should remain resident on the workstations for which they were intended. However, there are a number of applications which were developed which may be better suited for one of the hosts.

3.2.1 Display Sharing

Display Sharing is a new function being planned for the MCC to give the Flight Controllers the same functionality as the thumb wheel used to pull up any DTE display on the consoles. Display Sharing is the distribution of output information from applications on a source workstation to other workstations in the system. The MCC Display Sharing capability is divided into three categories: telecast; monitor attach; and display copy. Telecast allows a source workstation to request a distribution channel and then output to simultaneously display information for a single window to its local monitor and the channel allowing access to other workstation monitors. To receive the display information; a receiving workstation connects to a defined channel. Monitor Attach is basically an over-the-shoulder view of a workstation monitor. All information on the source workstation's monitor will appear on a receiving monitor and the receiving monitor will track the source workstation displays. Display copy is an image capture for a specific frame. Once captured, the image frame is distributed to a specific receiving workstation.

At this time, there is not enough information available to make a recommendation on the residency of this application.

3.2.2 Display Manager

The Display Manager manages the displays created by Display Builder during operation. It also uses Data Acquisition to access the shared data on the MOC. Since every workstation will have displays to be managed, it should reside on the workstation.

3.2.3 MUS

This interactive application is used by the flight controllers to request (through GDR) NRT data from the MOC. Due to its interactive nature and the possible need for NRT data displayed quickly, it is best to leave this application on the workstation.

3.2.4 Flight Notes

The Flight Notes function is an application which can be used to automatically write up the flight notes which are sometimes required during the course of a mission. This application is appropriately placed on the workstation since it has a highly interactive interface.

3.2.5 E-mail

The mail function currently resides on the workstations and requires a user to log onto a certain workstation to receive their mail. It should be more centralized so that a user can log onto any workstation in the network and receive mail.

3.3 Universal Functions

In any large system, there are functions which most of the users require to perform their job. The residency decision for an universal application can be very simple or very complicated. In the simple cases, low usage requirements, developmental applications, nonreal-time requirements, and local data requirements sway the decision heavily to either the workstation or the mainframe. In the complicated cases, the response time, data requirements, and frequency factors need to be evaluated collectively so that the most effective solution can be determined. Application of the residency questions for these types of applications will help to clear the picture in the complicated cases and verify the residency for the simpler cases. The MCC universal functions have been included in the sections which pertain to their current residency.

4.0 PLACEMENT OF NEWLY DEVELOPED APPLICATIONS

In order to prepare for future advancements in the workstation and distributed processing environment, a systems designer needs to develop software in a way which will lead to more efficient execution and resource utilization in the distributed system of the MCC. When new applications are being designed for a system which is already in operation, the designer should be familiar with the residency questions from Volume I so that all of the factors which will impact the efficiency of the final product on its intended target system will be acknowledged and incorporated into the requirements for the application. Once the requirements have been assigned, the principles discussed in Chapter Four of Volume I concerning the understandability and maintainability of software should be followed.

During the development of a new application, there are a large number of concerns which affect the residency of this software. These concerns include: 1) the types of functions which need to be performed; 2) maintainability of the software across the system; 3) the amount of user interaction; 4) the universal need for the application; and 5) the performance requirements of the application. All of these concerns are addressed by the residency questions. During the development stages of the application's components, there needs to be a methodology used to insure that all of the residency issues are considered for the particular application being developed. The residency questions should be used as a checklist to accomplish this task. The primary goal of these objectives is to design software which is verifiable and correct while at the same time controlling the complexity of the system.

The first step in the development of software for the cooperating mainframe/workstation environment is the decomposition of the software into processes which perform different functions (i.e. separate user interface functions from calculation functions). During this decomposition phase, the developer should then design the processes into modules which perform specific tasks and allow modules which perform common tasks to be reused by other applications. Once the requirements have been decomposed into specific processes and tasks, the ideas of task partitioning and task allocation discussed in Chapter Two of Volume I should also be used to better decompose the software into functions and modules which can then be partitioned to best utilize the system resources.

Once the applications are developed, a properly implemented configuration management system is a major necessity for the success of any software system. Configuration management is especially important in a distributed system to control the duplication of code, tasks, and functions and to insure that only validated software is executed during normal operation.

In summary, as new applications are conceptualized their design needs to be attacked by first using the residency questions as a checklist for examining the aspects of a cooperating environment which will impact the

C-2

residency, and therefore the design, of the application. Once these issues have been considered, they should weigh heavily on the decisions made during the implementation of the remaining design and development procedures. If a new application is designed and developed using the principles discussed above, determining the residency of a new application should be a fairly straight forward task because it will have been designed with the residency issues in mind.

5.0 CONCLUSIONS

The residency question of software in an environment moving from a centralized, time-sharing environment to a distributed processing environment using cooperating mainframes and engineering workstations is not an easy one to answer. This research has concentrated on discussing the factors associated with the coupling of workstations and mainframes. These discussions have tried to make the user think of all factors which impact the residency of an application between a mainframe and a workstation. It may seem trivial to attempt to answer all of the questions proposed, but it insures that the developer analyzes each of the contributing factors. Applying these questions can be compared to a checklist of items. Taken independently, each item may seem insignificant, but when considered as a whole, if one consideration is missed, it could have a significant impact on the outcome.

The best possible situation for a large, widely distributed, cooperative system would be to have a software engineering organization which is singularly responsible and knowledgeable of the effects outlined in the preceding chapters. The residence of each new entry into the system software vault should be considered totally against the overall operational effectiveness avoiding the biased and limited examinations of a single discipline. Determination of the residency of applications requires a coordination and understanding by the system designers and developers. Without coordination among the system designers and developers there will not be an opportunity to reuse code developed by other disciplines.

5.1 The Necessity of a Centralized Host

Throughout the course of this research, it has been acknowledged that there is still a need for the host/mainframe in the MCC environment. Although the workstations have approached the mainframes in their speed factors, the storage and reliability attained by the host is still an important asset to the environment. In the MCC's current configuration, the host/mainframes should continue to be used as the central processing and storage unit for space vehicle data which is needed by all of the mission support teams. It is also the most reasonable location for the storage of the Configuration Management data base and a few discipline-specific applications whose outputs are needed by other disciplines. One final benefit of the host/mainframes is their reliability. Both the RTH and the FSH are configured to have a backup system pickup their processing if they should ever go down. This is an important feature for integrity of data which is at times critical.

It can be foreseen that as workstations continue to grow, they will be able to pickup some of the processing currently executed on the host. However, for the system to run efficiently without host/mainframe support, there needs to be a communication mechanism which will allow the workstations at different disciplines to pass common data back and forth.

Until this type of communication is available in the MCC, the host/mainframes will still play a major role in the operation of the MCC.

5.2 Future Directions

In this final section, suggestions have been made for the future environment of the MCC. As the workstations advance and become less expensive and more powerful, the necessity for the mainframe/host may dwindle. The following topics discuss some alternative ideas which could be implemented in the MCC.

5.2.1 Centralized Control/Distributed Applications

In this report, applying workstations as a solution has basically been another way of saying distributed processing. Actually, another mode exists that has a workstation acting as a centralized application controller over a distributed process. A single workstation can take on the role of centralized logical control for a distributed process. The distributed process could actually be executing on the mainframe, among the workstations, or on both. One such possible situation, such as having a specific workstation acting as a file server, requires the workload to realize connections across the network. When N users, or clients, need to communicate with every other user, or server, then a fairly large number of connections need to be sustained between all of the processes. Another approach is to distribute the process, but centralize the control and the access between the distributed processes through a single logical point processor. In this case, a workstation can act as the logical center of a star network. Appearing as the client and server, it controls the whens, ifs, hows, and even the whats of every communication associated with that particular process. The most obvious advantage of having fewer logical connections, now N at the most, is offset by the possibility of incurring unsuitable delay while communicating to a destination. Some other features are the very rigid control of the application, data verification, timing monitoring, and synchronization possibilities. These possible trade-offs need to be measured carefully when assigning a workstation to the role of centralized controller in a distributed application.

5.2.2 Load Distribution

As workstations become more prominent in the MCC and the disciplines begin to develop more applications to run on them, consideration should be given to implementing a load distribution algorithm. This topic was discussed in Volume I, Chapter Five. Load distribution can be described as being either load sharing (LS) or load balancing (LB). LS attempts to conserve the ability of the system to perform work by assuring that no node within the system is idle while there exists a demand for service anywhere in the system. LB goes a step further by striving to equalize the entire workload among all of the nodes. LB can be further characterized by the static or dynamic nature of its control. Static control only deals with the initial placement of a process. Dynamic control employs a migration

component capable of transferring a process once it has already started to execute.

These algorithms have been implemented in distributed computing systems to improve the performance of the system by efficiently utilizing the computing power of the entire system. By using remote execution utilities the workstation-based distributed system can transfer jobs from heavily loaded nodes to inactive or less-busy nodes. This leads to an effective means of maximizing the computing power within a distributed system.

Although a communication delay must be incurred by transferring a job from one node to another, the performance of a distributed computing system is generally improved by an effective load balancing policy. Implementing a load distribution algorithm in the MCC environment may not be necessary at this point in time, but eventually, as more demands are put onto the workstations, the need will arise.

APPENDIX A: ACRONYMS & DEFINITIONS

AEG	Analytic Ephemeris Generator
AOS	Acquisition of Signal
CCS/C	Command and Control System/Control
CPU	Central Processing Unit
CHC	Channel Capacity
CF	Clock Frequency
CISC	Complex Instruction Set Computer
CM	Configuration Management
COMSERVER	Communications Server
COTS	Commercial Off-the-Shelf
DASD	Direct Access Storage Device
DCL	Data Control List
DBMS	Data Base Management System
DDHS	Dump Data Handling System
DNP	Dedicated Numerics Processor
DPS	Data Processing Support
DSC	Dynamic Standby Computer
DTE	Digital Television Equipment
EECOM	Electrical, Environ, and Consumables
FAP	Failure Analysis Program
FDO	Flight Dynamics Officer
FLOPS	Floating Point Operations per Second
FSH	Flight Support Host
FPR	Floating Point Register
GC	Ground Control
GDR	Generalized Data Retrieval
GMT	Greenwich Mean Time
GNC	Guidance, Navigation, and Control
GPAMP	General Purpose Attached Math Processor
GPR	General Purpose Register
GSTDN	Ground Spaceflight Tracking and Data Network
H/I	Hardware Interruption
IDA	Intermediate Data Array
IGP	Integral Graphics Processor
I/O/I	Input/Output Interruption
IPCL	Instrumentation Parts and Components List
IPS	Instrument Pointing System
IV	Independent Verification
LB	Load Balancing
LRU	Line Replaceable Unit
LS	Load Sharing
LAN	Local Area Network
MB	Megabytes
MBPS	Megabytes per Second
MCC	Mission Control Center
MCCU	Mission Control Center Upgrade
MED	Manual Entry Device
MEPS	Million Executions per Second
MET	Mission Elapsed Time

MFLOPS	Million Floating-point Operations per Second
MHz	Megahertz
MIPS	Million Instructions per Second
MITS	MOD IPS TACAN Subsystem
MMC	Main Memory Capacity
MOC	Mission Operations Computer
MOD	Mission Operations Directorate
MP	Multiple Processors
MRC	Maximum Rate of Computation
MSID	Measurement Stimulus Identification
MTM	MOC Telemetry Message
MVS	Multiple Virtual Storage
NCH	Number of Channels
NCIC	Network Communications Interface Common
NDD	Network Data Driver
NOM	Network Output Multiplexer
NRT	Near Real-time Telemetry
NSS	Network Support Software
OS/I	Operating System Interruption
OST	Operations Support Team
PBI	Push-button Indicator
PDT	Payload Data Tape
P/I	Program Interruption
P/L	Payload
PPD	Projection Plotting Display
PPL	Processed Parameter List
RISC	Reduced Instruction Set Computer
RTH	Real-time Host
SAMU	Smallest Addressable Memory Unit
SAT	Satellite Acquisition Table
Sc	Speed of the Computer
SCAP	Shuttle Configuration Analysis Program
SDT	Shuttle Data Tape
SIM	Simulation
SMP	Special Math Processor
SMC	Standard Math Coprocessor
S/S	System Software
STS	Space Transportation System
TDR	Trajectory Data Retrieval
TDRSS	Tracking and Data Relay Satellite System
TPC	Telemetry Preprocessing Computer
TSO	Time-Share Option
VS	Virtual Storage
WEX	Workstation Executive
W/S	Workstation

APPENDIX B: RESIDENCY QUESTION MATRIX

Appendix B contains System Summary Information, answers to System Specific Questions and a series of matrices containing answers to the residency questions presented in Volume I for each of the applications discussed in Chapter 3 of Volume II.

The question checklist utilized was extracted from the concluding chapter of Volume I. Using available information, these questions have been applied to those host and workstation applications which are candidates for migration. The questions are arranged so that if a "yes" answer results, then increased weight to the host as the residence is indicated. If a "no" answer is the result, then the engineering workstation weighing is increased. Questions marked with a single asterisk indicate that failure on these threshold items require no less than host computing power level. Items marked with a double asterisk indicate that the host can not meet the requirements in its aggregated operational status requiring off loading of an application onto the workstation. For connection to the body of this report, each question is referenced to the appropriate chapter section number in Volume I. This number can be used to locate the research discussion. The application reference number refers to the residency discussion associated with the application in Volume II.

These general criteria questions will build a weighing factor that will hopefully indicate a trend for either the host or the workstation for the residency of the software. The questions that do not have asterisks should be considered equal in influence on the decision. A reasonable, if not somewhat arbitrary, rule would say that if 70 percent of the questions are answered "no" then workstation residency is the decision. If 70 percent are answered "yes" then host residency is the most likely answer. Between these two limits, would be considered a range of optional consideration.

System Information Summary

	IBM 3081	MASSCOMP 6650
1) SPEED FACTORS		
MRC	N/A	N/A
MIPS	15.6	13.3
MFLOPS	2.1	.77
CF	41.67 MHz	33.0 MHz

$$Sc = ((Wc*MFLOPS + Wo*MIPS)*MRC)^{.5}$$

where:

Wc = the fraction of time spent in mathematical computations
Wo = the fraction of time spent in other operations

Note: MRC is not available for these computations

	IBM 3081	MASSCOMP 6650	RATIO
Wc = 0 Wo = 1	15.6 MEPS	13.3 MEPS	1.17
Wc = .2 Wo = .8	12.9 MEPS	10.79 MEPS	1.19
Wc = .4 Wo = .6	10.2 MEPS	8.29 MEPS	1.23
Wc = .5 Wo = .5	8.85 MEPS	7.03 MEPS	1.26
Wc = .6 Wo = .4	7.5 MEPS	5.78 MEPS	1.30
Wc = .8 Wo = .2	4.8 MEPS	3.28 MEPS	1.46
Wc = 1 Wo = 0	2.1 MEPS	.77 MEPS	2.73

	IBM 3081	MASSCOMP 6650
2) CAPACITY FACTORS MMC	16 MB	32 MB
3) CPU RATINGS	70	55
4) ROBUSTNESS PRODUCT	.970	.865

System Specific Questions

1.	What type of operating system is being used on the host/mainframe? (5.1.1)	**Centralized**
2.	What type of operating system is being used on the workstation? (5.1.1)	**Localized**
3.	Is there a load distribution algorithm for the system? (5.1.2)	**No**
4.	If there is load distribution, is it load sharing? (5.1.2)	**N/A**
5.	If there is load distribution, is it load balancing? (5.1.2)	**N/A**
6.	Is there shared data on the host/mainframe? (5.3)	**Yes**
7.	Is there shared data on the workstations? (5.3)	**No**
8.	Is there a distributed file system? (5.3)	**No**
9.	Is the robustness factor of the host greater than .8? (3.3.4)	**Yes**
10.	Is the robustness factor of the workstation less than .8? (3.3.4)	**Yes**

Application	3.1.1		3.1.2		3.1.3		3.1.4		3.1.5	
Computing Power	Y	N	Y	N	Y	N	Y	N	Y	N
1. Is the host Sc 64 times greater than the workstation Sc? (3.3.1)		X		X		X		X		X
2. Is the host Sc 256 times greater than the workstation Sc? (3.3.1)		X		X		X		X		X
3. Is the host Sc 1024 times greater than the workstation Sc? (3.3.1)		X		X		X		X		X
4. Is the workstation value of MIPS available exceeded by the application MIPS requirement? (3.3.2)*				X		X		X		
5. Is the aggregated application required value of MIPS exceeded by the host MIPS value? (3.3.2)**				X		X		X		
6. Is the workstation value of MIPS exceeded by the aggregated application requirements? (3.3.2)*				X		X		X		
7. Are the aggregated application requirements of MIPS with the addition of this application exceeded by the MIPS value of the host? (3.3.2)**										
8. Is the workstation value of MFLOPS available exceeded by the application requirement? (3.3.2)*										
9. Is the application required value of MFLOPS exceeded by the MFLOPS available on the host? (3.3.2)**										
10. Is the workstation MFLOPS value available exceeded by the aggregated requirements with the addition of this application? (3.3.2)*										
11. Is the aggregated requirement of the application program MFLOPS exceeded by the host value of MFLOPS available? (3.3.2)**										

Application	3.1.1		3.1.2		3.1.3		3.1.4		3.1.5	
Computing Power (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
12. Is the maximum acceptable response time for the software less than 10 msec? (3.2.1)		X		X		X		X		
13. Is the maximum acceptable response time for the software less than 100 msec? (3.2.1)		X		X		X		X		
14. Is the maximum acceptable response time for the software less than 1.0 sec? (3.2.1)	X			X			X			
15. Is the maximum acceptable response time for the software less than 10.0 sec? (3.2.1)	X			X			X			
16. Is the required CPU rating estimated to be higher than the estimated workstation CPU rating? (3.3.3)*										
17. Is the host CPU rating estimated to be higher than the estimated application required CPU rating? (3.3.3)**										
18. Is the required MMC greater than 50 percent of the workstation MMC that is available? (3.2.3.1.1)				X		X				
19. Is the required MMC greater than the workstation MMC that is available? (3.2.3.1.1)*				X		X				
20. Is the available host MMC greater than the required application MMC? (3.2.3.1.1)**			X		X					
21. Is the ratio of estimated number of computations compared to the number of other instruction executions greater than 100? (3.3.1)										
22. Is the ratio of estimated number of computations compared to the number of other instruction executions greater than 100,000? (3.3.1)										

Application	3.1.1		3.1.2		3.1.3		3.1.4		3.1.5	
Software Development Issues	Y	N	Y	N	Y	N	Y	N	Y	N
1. Is the application primarily a noninteractive function? (4.2)	X		X		X		X		X	
2. If there is user interaction, is it only front-end queries? (4.2)		-		-	-	-		-		-
3. If there are queries, can they be separated from the calculations and passed as parameters to the calculation process? (4.2)		-		-		-		-		-
4. Do many disciplines require this application's function? (4.2)	X		X		X		X			X
5. Is the application only needed under special circumstances? (4.3)		X		X	X			X	X	
6. Is the application a development tool? (4.3)		X		X		X		X		X
7. Is the response time requirement critical? (4.3)	X			X	X		X		X	
8. Are there host applications which will need this application's output? (4.4)	X		X			X	X		X	
9. Is there one other discipline which will use the application's output? (4.4)	X		X			X	X		X	
10. Are there five disciplines which will use the application's output? (4.4)	X		X			X	X		X	
11. Do all of the disciplines use the application's output? (4.4)	X		X			X		X		X

Application	3.1.1		3.1.2		3.1.3		3.1.4		3.1.5	
Control Considerations	Y	N	Y	N	Y	N	Y	N	Y	N
1. Can the application run in the background? (5.1.2)	X		X		X		X		X	
2. Does the application have communication needs with applications on the host/mainframe? (5.2)		X	X		X				X	
3. Is the application independent of applications on the workstation? (5.2)	X		X		X		X		X	
4. Is the application dependent on applications on the host? (5.2)		X	X		X			X	X	
5. If there are communication needs, does the application have infrequent communication needs with applications on the host/mainframe? (5.2)		-	X		X			X	X	
6. If there are communication needs, does the application have periodic communication needs with applications on the host/mainframe? (5.2)		-	X		X			X	X	
7. If there are communication needs, does the application have heavy communication needs with applications on the host/mainframe? (5.2)		-	X		X			X		X
8. Does the application have infrequent communication needs with more than one workstation? (5.2)		X		X	X		X			X
9. Does the application have periodic communication needs with more than one workstation? (5.2)		X		X	X		X			X
10. Does the application have heavy communication needs with more than one workstation? (5.2)		X		X		X		X		X

Application	3.1.1		3.1.2		3.1.3		3.1.4		3.1.5	
Control Considerations (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
11. Is the execution time a critical factor? (5.2)	X		X			X	X		X	
12. Does the application require shared data access? (5.3)	X		X		X		X		X	
13. Does the application access the shared data frequently? (5.3)	X		X		X		X		X	
14. Does the application modify shared data? (5.3)	X		X		X		X		X	

Application	3.1.1		3.1.2		3.1.3		3.1.4		3.1.5	
Networking Delay Impact	Y	N	Y	N	Y	N	Y	N	Y	N
1. Are there more than eight message types required to be sent or received within this application? (6.4.2)										
2. Are there more than 32 message types required to be sent or received within this application? (6.4.2)										
3. Are the average message lengths greater than 100 bytes? (6.4.2)										
4. Are the average message lengths greater than 1000 bytes? (6.4.2)										
5. Are more than 10 messages per minute expected to be sent for this application? (6.4.2)										
6. Are more than 1000 messages per minute expected to be sent for this application? (6.4.2)										
7. Is the hardware network transmission speed less than 1 Megabit per second? (6.2)										
8. Is the hardware network transmission speed less than 10 Megabits per second? (6.2)										
9. Is the full protocol used to communicate at each level of the OSI model? (6.3)										
10. Is the BER of the network estimated to be worse than one bit in 100,000? (6.4.1)										
11. Is the BER of the network estimated to be worse than one bit in ten million? (6.4.1)										

Application	3.1.1		3.1.2		3.1.3		3.1.4		3.1.5	
Networking Delay Impact (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
12. Does this application program have data messages for more than eight user destinations? (6.4.2)										
13. Does this application program have data messages for more than 32 user destinations? (6.4.2)										
14. Does this application have more than eight users who can make data inquiries? (6.4.2)	X			X	X		X		X	
15. Does this application have more than 32 users who can make data inquiries? (6.4.2)	X			X	X			X		X
16. Is the network a connectionless service? (6.4.2.2)										
17. Is the network utilization rate expected to be above 50 percent after adding this application? (6.4.2.2)										
18. Is the greatest peak communication demand of this program above 10 percent of the network transmission speed? (6.4.2.2)										

Application	3.1.6		3.1.7		3.1.8		3.1.9		3.1.10	
Computing Power	Y	N	Y	N	Y	N	Y	N	Y	N
1. Is the host Sc 64 times greater than the workstation Sc? (3.3.1)		X		X		X		X		X
2. Is the host Sc 256 times greater than the workstation Sc? (3.3.1)		X		X		X		X		X
3. Is the host Sc 1024 times greater than the workstation Sc? (3.3.1)		X		X		X		X		X
4. Is the workstation value of MIPS available exceeded by the application MIPS requirement? (3.3.2)*										
5. Is the aggregated application required value of MIPS exceeded by the host MIPS value? (3.3.2)**										
6. Is the workstation value of MIPS exceeded by the aggregated application requirements? (3.3.2)*										
7. Are the aggregated application requirements of MIPS with the addition of this application exceeded by the MIPS value of the host? (3.3.2)**										
8. Is the workstation value of MFLOPS available exceeded by the application requirement? (3.3.2)*										
9. Is the application required value of MFLOPS exceeded by the MFLOPS available on the host? (3.3.2)**										
10. Is the workstation MFLOPS value available exceeded by the aggregated requirements with the addition of this application? (3.3.2)*										
11. Is the aggregated requirement of the application program MFLOPS exceeded by the host value of MFLOPS available? (3.3.2)**										

Application	3.1.6		3.1.7		3.1.8		3.1.9		3.1.10	
Computing Power (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
12. Is the maximum acceptable response time for the software less than 10 msec? (3.2.1)		X		X		X		X		
13. Is the maximum acceptable response time for the software less than 100 msec? (3.2.1)		X		X		X		X		
14. Is the maximum acceptable response time for the software less than than 1.0 sec? (3.2.1)	X			X	X		X		X	
15. Is the maximum acceptable response time for the software less than 10.0 sec? (3.2.1)	X			X	X		X		X	
16. Is the required CPU rating estimated to be higher than the estimated workstation CPU rating? (3.3.3)*										
17. Is the host CPU rating estimated to be higher than the estimated application required CPU rating? (3.3.3)**										
18. Is the required MMC greater than 50 percent of the workstation MMC that is available? (3.2.3.1.1)										
19. Is the required MMC greater than the workstation MMC that is available? (3.2.3.1.1)*										
20. Is the available host MMC greater than the required application MMC? (3.2.3.1.1)**										
21. Is the ratio of estimated number of computations compared to the number of other instruction executions greater than 100? (3.3.1)										
22. Is the ratio of estimated number of computations compared to the number of other instruction executions greater than 100,000? (3.3.1)										

Application	3.1.6		3.1.7		3.1.8		3.1.9		3.1.10	
Software Development Issues	Y	N	Y	N	Y	N	Y	N	Y	N
1. Is the application primarily a noninteractive function? (4.2)	X		X		X		X		X	
2. If there is user interaction, is it only front-end queries? (4.2)		-		-	-	-	-		-	
3. If there are queries, can they be separated from the calculations and passed as parameters to the calculation process? (4.2)		-		-		-	-		-	
4. Do many disciplines require this application's function? (4.2)		X	X			-	X		X	
5. Is the application only needed under special circumstances? (4.3)	X		X			X	X			X
6. Is the application a development tool? (4.3)		X		X		X		X		X
7. Is the response time requirement critical? (4.3)		X		X	X		X		X	
8. Are there host applications which will need this application's output? (4.4)	X		X			X	X		X	
9. Is there one other discipline which will use the application's output? (4.4)	X		X		X		X			X
10. Are there five disciplines which will use the application's output? (4.4)	X		X			X	X			X
11. Do all of the disciplines use the application's output? (4.4)		X		X		X	X			X

Application	3.1.6		3.1.7		3.1.8		3.1.9		3.1.10	
Control Considerations	Y	N	Y	N	Y	N	Y	N	Y	N
1. Can the application run in the background? (5.1.2)	X		X		X		X		X	
2. Does the application have communication needs with applications on the host/mainframe? (5.2)	X		X		X			X	X	
3. Is the application independent of applications on the workstation? (5.2)	X		X		X			X		X
4. Is the application dependent on applications on the host? (5.2)	X		X		X			X	X	
5. If there are communication needs, does the application have infrequent communication needs with applications on the host/mainframe? (5.2)	X			X	X				X	
6. If there are communication needs, does the application have periodic communication needs with applications on the host/mainframe? (5.2)	X			X	X				X	
7. If there are communication needs, does the application have heavy communication needs with applications on the host/mainframe? (5.2)		X		X		X			X	
8. Does the application have infrequent communication needs with more than one workstation? (5.2)		X		X	X		X		X	
9. Does the application have periodic communication needs with more than one workstation? (5.2)		X		X	X		X		X	
10. Does the application have heavy communication needs with more than one workstation? (5.2)		X		X	X		X		X	

Application	3.1.6		3.1.7		3.1.8		3.1.9		3.1.10	
Control Considerations (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
11. Is the execution time a critical factor? (5.2)		X		X	X		X		X	
12. Does the application require shared data access? (5.3)	X		X			X		X	X	
13. Does the application access the shared data frequently? (5.3)	X		X			X		X	X	
14. Does the application modify shared data? (5.3)	X			X		X		X		X

Application	3.1.6		3.1.7		3.1.8		3.1.9		3.1.10	
Networking Delay Impact	Y	N	Y	N	Y	N	Y	N	Y	N
1. Are there more than eight message types required to be sent or received within this application? (6.4.2)										
2. Are there more than 32 message types required to be sent or received within this application? (6.4.2)										
3. Are the average message lengths greater than 100 bytes? (6.4.2)										
4. Are the average message lengths greater than 1000 bytes? (6.4.2)										
5. Are more than 10 messages per minute expected to be sent for this application? (6.4.2)										
6. Are more than 1000 messages per minute expected to be sent for this application? (6.4.2)										
7. Is the hardware network transmission speed less than 1 Megabit per second? (6.2)										
8. Is the hardware network transmission speed less than 10 Megabits per second? (6.2)										
9. Is the full protocol used to communicate at each level of the OSI model? (6.3)										
10. Is the BER of the network estimated to be worse than one bit in 100,000? (6.4.1)										
11. Is the BER of the network estimated to be worse than one bit in ten million? (6.4.1)										

Application	3.1.6		3.1.7		3.1.8		3.1.9		3.1.10	
Networking Delay Impact (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
12. Does this application program have data messages for more than eight user destinations? (6.4.2)							X		X	
13. Does this application program have data messages for more than 32 user destinations? (6.4.2)							X		X	
14. Does this application have more than eight users who can make data inquiries? (6.4.2)	X		X			X	X		X	
15. Does this application have more than 32 users who can make data inquiries? (6.4.2)		X		X		X	X		X	
16. Is the network a connectionless service? (6.4.2.2)										
17. Is the network utilization rate expected to be above 50 percent after adding this application? (6.4.2.2)										
18. Is the greatest peak communication demand of this program above 10 percent of the network transmission speed? (6.4.2.2)										

Application	3.1.11		3.1.12		3.1.13		3.1.14		3.1.15	
Computing Power	Y	N	Y	N	Y	N	Y	N	Y	N
1. Is the host Sc 64 times greater than the workstation Sc? (3.3.1)		X		X		X		X		X
2. Is the host Sc 256 times greater than the workstation Sc? (3.3.1)		X		X		X		X		X
3. Is the host Sc 1024 times greater than the workstation Sc? (3.3.1)		X		X		X		X		X
4. Is the workstation value of MIPS available exceeded by the application MIPS requirement? (3.3.2)*		X		X		X				
5. Is the aggregated application required value of MIPS exceeded by the host MIPS value? (3.3.2)**										
6. Is the workstation value of MIPS exceeded by the aggregated application requirements? (3.3.2)*										
7. Are the aggregated application requirements of MIPS with the addition of this application exceeded by the MIPS value of the host? (3.3.2)**										
8. Is the workstation value of MFLOPS available exceeded by the application requirement? (3.3.2)*										
9. Is the application required value of MFLOPS exceeded by the MFLOPS available on the host? (3.3.2)**										
10. Is the workstation MFLOPS value available exceeded by the aggregated requirements with the addition of this application? (3.3.2)*										
11. Is the aggregated requirement of the application program MFLOPS exceeded by the host value of MFLOPS available? (3.3.2)**										

Application	3.1.11		3.1.12		3.1.13		3.1.14		3.1.15	
Computing Power (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
12. Is the maximum acceptable response time for the software less than 10 msec? (3.2.1)				X		X				
13. Is the maximum acceptable response time for the software less than 100 msec? (3.2.1)				X		X				
14. Is the maximum acceptable response time for the software less than 1.0 sec? (3.2.1)				X		X				
15. Is the maximum acceptable response time for the software less than 10.0 sec? (3.2.1)				X		X				
16. Is the required CPU rating estimated to be higher than the estimated workstation CPU rating? (3.3.3)*										
17. Is the host CPU rating estimated to be higher than the estimated application required CPU rating? (3.3.3)**										
18. Is the required MMC greater than 50 percent of the workstation MMC that is available? (3.2.3.1.1)		X		X						
19. Is the required MMC greater than the workstation MMC that is available? (3.2.3.1.1)*		X		X						
20. Is the available host MMC greater than the required application MMC? (3.2.3.1.1)**	X		X							
21. Is the ratio of estimated number of computations compared to the number of other instruction executions greater than 100? (3.3.1)										
22. Is the ratio of estimated number of computations compared to the number of other instruction executions greater than 100,000? (3.3.1)										

Application	3.1.11		3.1.12		3.1.13		3.1.14		3.1.15	
Software Development Issues	Y	N	Y	N	Y	N	Y	N	Y	N
1. Is the application primarily a noninteractive function? (4.2)	X		X		X		X		X	
2. If there is user interaction, is it only front-end queries? (4.2)	X			-	-	X			-	
3. If there are queries, can they be separated from the calculations and passed as parameters to the calculation process? (4.2)		X		-		X			-	
4. Do many disciplines require this application's function? (4.2)		X	X		X		X			X
5. Is the application only needed under special circumstances? (4.3)	X		X		X		X			
6. Is the application a development tool? (4.3)		X	X			X		X		X
7. Is the response time requirement critical? (4.3)	X			X		X		X		
8. Are there host applications which will need this application's output? (4.4)	X			X		X		X		
9. Is there one other discipline which will use the application's output? (4.4)		X	X		X		X			
10. Are there five disciplines which will use the application's output? (4.4)		X	X		X		X			
11. Do all of the disciplines use the application's output? (4.4)		X	X		X			X		

Application	3.1.11		3.1.12		3.1.13		3.1.14		3.1.15	
Control Considerations	Y	N	Y	N	Y	N	Y	N	Y	N
1. Can the application run in the background? (5.1.2)	X		X			X		X	X	
2. Does the application have communication needs with applications on the host/mainframe? (5.2)	X			X	X			X		
3. Is the application independent of applications on the workstation? (5.2)	X		X		X		X		X	
4. Is the application dependent on applications on the host? (5.2)	X			X		X		X		
5. If there are communication needs, does the application have infrequent communication needs with applications on the host/mainframe? (5.2)	X			X		X				
6. If there are communication needs, does the application have periodic communication needs with applications on the host/mainframe? (5.2)	X			X		X				
7. If there are communication needs, does the application have heavy communication needs with applications on the host/mainframe? (5.2)	X			X		X				
8. Does the application have infrequent communication needs with more than one workstation? (5.2)		X	X			X		X		
9. Does the application have periodic communication needs with more than one workstation? (5.2)		X	X			X		X		
10. Does the application have heavy communication needs with more than one workstation? (5.2)		X		X		X		X		

Application	3.1.11		3.1.12		3.1.13		3.1.14		3.1.15	
Control Considerations (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
11. Is the execution time a critical factor? (5.2)		X		X		X		X		
12. Does the application require shared data access? (5.3)		X		X		X		X		X
13. Does the application access the shared data frequently? (5.3)		X		X		X		X		X
14. Does the application modify shared data? (5.3)		X		X		X		X		X

Application	3.1.11		3.1.12		3.1.13		3.1.14		3.1.15	
Networking Delay Impact	Y	N	Y	N	Y	N	Y	N	Y	N
1. Are there more than eight message types required to be sent or received within this application? (6.4.2)										
2. Are there more than 32 message types required to be sent or received within this application? (6.4.2)										
3. Are the average message lengths greater than 100 bytes? (6.4.2)										
4. Are the average message lengths greater than 1000 bytes? (6.4.2)										
5. Are more than 10 messages per minute expected to be sent for this application? (6.4.2)										
6. Are more than 1000 messages per minute expected to be sent for this application? (6.4.2)										
7. Is the hardware network transmission speed less than 1 Megabit per second? (6.2)										
8. Is the hardware network transmission speed less than 10 Megabits per second? (6.2)										
9. Is the full protocol used to communicate at each level of the OSI model? (6.3)										
10. Is the BER of the network estimated to be worse than one bit in 100,000? (6.4.1)										
11. Is the BER of the network estimated to be worse than one bit in ten million? (6.4.1)										

Application	3.1.11		3.1.12		3.1.13		3.1.14		3.1.15	
Networking Delay Impact (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
12. Does this application program have data messages for more than eight user destinations? (6.4.2)		X								
13. Does this application program have data messages for more than 32 user destinations? (6.4.2)		X								
14. Does this application have more than eight users who can make data inquiries? (6.4.2)		X								
15. Does this application have more than 32 users who can make data inquiries? (6.4.2)		X								
16. Is the network a connectionless service? (6.4.2.2)										
17. Is the network utilization rate expected to be above 50 percent after adding this application? (6.4.2.2)										
18. Is the greatest peak communication demand of this program above 10 percent of the network transmission speed? (6.4.2.2)										

Application	3.1.16		3.1.17		3.2.1		3.2.2		3.2.3	
Computing Power	Y	N	Y	N	Y	N	Y	N	Y	N
1. Is the host Sc 64 times greater than the workstation Sc? (3.3.1)		X		X		X		X		X
2. Is the host Sc 256 times greater than the workstation Sc? (3.3.1)		X		X		X		X		X
3. Is the host Sc 1024 times greater than the workstation Sc? (3.3.1)		X		X		X		X		X
4. Is the workstation value of MIPS available exceeded by the application MIPS requirement? (3.3.2)*										
5. Is the aggregated application required value of MIPS exceeded by the host MIPS value? (3.3.2)**										
6. Is the workstation value of MIPS exceeded by the aggregated application requirements? (3.3.2)*										
7. Are the aggregated application requirements of MIPS with the addition of this application exceeded by the MIPS value of the host? (3.3.2)**										
8. Is the workstation value of MFLOPS available exceeded by the application requirement? (3.3.2)*										
9. Is the application required value of MFLOPS exceeded by the MFLOPS available on the host? (3.3.2)**										
10. Is the workstation MFLOPS value available exceeded by the aggregated requirements with the addition of this application? (3.3.2)*										
11. Is the aggregated requirement of the application program MFLOPS exceeded by the host value of MFLOPS available? (3.3.2)**										

Application	3.1.16		3.1.17		3.2.1		3.2.2		3.2.3	
Computing Power (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
12. Is the maximum acceptable response time for the software less than 10 msec? (3.2.1)		X		X		X		X		X
13. Is the maximum acceptable response time for the software less than 100 msec? (3.2.1)		X		X		X		X		X
14. Is the maximum acceptable response time for the software less than 1.0 sec? (3.2.1)		X	X		X		X			X
15. Is the maximum acceptable response time for the software less than 10.0 sec? (3.2.1)			X		X		X		X	
16. Is the required CPU rating estimated to be higher than the estimated workstation CPU rating? (3.3.3)*										
17. Is the host CPU rating estimated to be higher than the estimated application required CPU rating? (3.3.3)**										
18. Is the required MMC greater than 50 percent of the workstation MMC that is available? (3.2.3.1.1)										
19. Is the required MMC greater than the workstation MMC that is available? (3.2.3.1.1)*										
20. Is the available host MMC greater than the required application MMC? (3.2.3.1.1)**										
21. Is the ratio of estimated number of computations compared to the number of other instruction executions greater than 100? (3.3.1)										
22. Is the ratio of estimated number of computations compared to the number of other instruction executions greater than 100,000? (3.3.1)										

Application	3.1.16		3.1.17		3.2.1		3.2.2		3.2.3	
Software Development Issues	Y	N	Y	N	Y	N	Y	N	Y	N
1. Is the application primarily a noninteractive function? (4.2)	X		X			X	X			X
2. If there is user interaction, is it only front-end queries? (4.2)		-		-	-	X		-	X	
3. If there are queries, can they be separated from the calculations and passed as parameters to the calculation process? (4.2)		-		-		X		-	-	
4. Do many disciplines require this application's function? (4.2)	X		X		X		X		X	
5. Is the application only needed under special circumstances? (4.3)		X		X	X			X	X	
6. Is the application a development tool? (4.3)		X		X		X		X		X
7. Is the response time requirement critical? (4.3)			X		X		X			X
8. Are there host applications which will need this application's output? (4.4)			X			X		X		X
9. Is there one other discipline which will use the application's output? (4.4)	X		X		X			X		X
10. Are there five disciplines which will use the application's output? (4.4)	X		X		X			X		X
11. Do all of the disciplines use the application's output? (4.4)		X	X			X		X		X

Application	3.1.16		3.1.17		3.2.1		3.2.2		3.2.3	
Control Considerations	Y	N	Y	N	Y	N	Y	N	Y	N
1. Can the application run in the background? (5.1.2)	X		X			X	X			X
2. Does the application have communication needs with applications on the host/mainframe? (5.2)		X		X		X		X	X	
3. Is the application independent of applications on the workstation? (5.2)	X		X			X		X	X	
4. Is the application dependent on applications on the host? (5.2)		X		X		X		X	X	
5. If there are communication needs, does the application have infrequent communication needs with applications on the host/mainframe? (5.2)		-		-		X	X		X	
6. If there are communication needs, does the application have periodic communication needs with applications on the host/mainframe? (5.2)		-		-		X	X		X	
7. If there are communication needs, does the application have heavy communication needs with applications on the host/mainframe? (5.2)		-		-		X	X		X	
8. Does the application have infrequent communication needs with more than one workstation? (5.2)		X		X	X			X		X
9. Does the application have periodic communication needs with more than one workstation? (5.2)		X		X	X			X		X
10. Does the application have heavy communication needs with more than one workstation? (5.2)		X		X	X			X		X

Application	3.1.16		3.1.17		3.2.1		3.2.2		3.2.3	
Control Considerations (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
11. Is the execution time a critical factor? (5.2)	X		X		X		X			X
12. Does the application require shared data access? (5.3)	X		X		X		X		X	
13. Does the application access the shared data frequently? (5.3)	X		X		X		X		X	
14. Does the application modify shared data? (5.3)	X		X			X		X		X

Application	3.1.16		3.1.17		3.2.1		3.2.2		3.2.3	
Networking Delay Impact	Y	N	Y	N	Y	N	Y	N	Y	N
1. Are there more than eight message types required to be sent or received within this application? (6.4.2)										
2. Are there more than 32 message types required to be sent or received within this application? (6.4.2)										
3. Are the average message lengths greater than 100 bytes? (6.4.2)										
4. Are the average message lengths greater than 1000 bytes? (6.4.2)										
5. Are more than 10 messages per minute expected to be sent for this application? (6.4.2)										
6. Are more than 1000 messages per minute expected to be sent for this application? (6.4.2)										
7. Is the hardware network transmission speed less than 1 Megabit per second? (6.2)										
8. Is the hardware network transmission speed less than 10 Megabits per second? (6.2)										
9. Is the full protocol used to communicate at each level of the OSI model? (6.3)										
10. Is the BER of the network estimated to be worse than one bit in 100,000? (6.4.1)										
11. Is the BER of the network estimated to be worse than one bit in ten million? (6.4.1)										

Application	3.1.16		3.1.17		3.2.1		3.2.2		3.2.3	
Networking Delay Impact (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
12. Does this application program have data messages for more than eight user destinations? (6.4.2)										
13. Does this application program have data messages for more than 32 user destinations? (6.4.2)										
14. Does this application have more than eight users who can make data inquiries? (6.4.2)										
15. Does this application have more than 32 users who can make data inquiries? (6.4.2)										
16. Is the network a connectionless service? (6.4.2.2)										
17. Is the network utilization rate expected to be above 50 percent after adding this application? (6.4.2.2)										
18. Is the greatest peak communication demand of this program above 10 percent of the network transmission speed? (6.4.2.2)										

Application	3.2.4		3.2.5							
Computing Power	Y	N	Y	N	Y	N	Y	N	Y	N
1. Is the host Sc 64 times greater than the workstation Sc? (3.3.1)		X		X						
2. Is the host Sc 256 times greater than the workstation Sc? (3.3.1)		X		X						
3. Is the host Sc 1024 times greater than the workstation Sc? (3.3.1)		X		X						
4. Is the workstation value of MIPS available exceeded by the application MIPS requirement? (3.3.2)*										
5. Is the aggregated application required value of MIPS exceeded by the host MIPS value? (3.3.2)**										
6. Is the workstation value of MIPS exceeded by the aggregated application requirements? (3.3.2)*										
7. Are the aggregated application requirements of MIPS with the addition of this application exceeded by the MIPS value of the host? (3.3.2)**										
8. Is the workstation value of MFLOPS available exceeded by the application requirement? (3.3.2)*										
9. Is the application required value of MFLOPS exceeded by the MFLOPS available on the host? (3.3.2)**										
10. Is the workstation MFLOPS value available exceeded by the aggregated requirements with the addition of this application? (3.3.2)*										
11. Is the aggregated requirement of the application program MFLOPS exceeded by the host value of MFLOPS available? (3.3.2)**										

Application	3.2.4		3.2.5							
Computing Power (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
12. Is the maximum acceptable response time for the software less than 10 msec? (3.2.1)		X		X						
13. Is the maximum acceptable response time for the software less than 100 msec? (3.2.1)		X		X						
14. Is the maximum acceptable response time for the software less than 1.0 sec? (3.2.1)		X		X						
15. Is the maximum acceptable response time for the software less than 10.0 sec? (3.2.1)		X		X						
16. Is the required CPU rating estimated to be higher than the estimated workstation CPU rating? (3.3.3)*										
17. Is the host CPU rating estimated to be higher than the estimated application required CPU rating? (3.3.3)**										
18. Is the required MMC greater than 50 percent of the workstation MMC that is available? (3.2.3.1.1)										
19. Is the required MMC greater than the workstation MMC that is available? (3.2.3.1.1)*										
20. Is the available host MMC greater than the required application MMC? (3.2.3.1.1)**										
21. Is the ratio of estimated number of computations compared to the number of other instruction executions greater than 100? (3.3.1)										
22. Is the ratio of estimated number of computations compared to the number of other instruction executions greater than 100,000? (3.3.1)										

Application	3.2.4		3.2.5							
Software Development Issues	Y	N	Y	N	Y	N	Y	N	Y	N
1. Is the application primarily a noninteractive function? (4.2)		X		X						
2. If there is user interaction, is it only front-end queries? (4.2)		X		X						
3. If there are queries, can they be separated from the calculations and passed as parameters to the calculation process? (4.2)		X		X						
4. Do many disciplines require this application's function? (4.2)	X		X							
5. Is the application only needed under special circumstances? (4.3)	X		X							
6. Is the application a development tool? (4.3)		X		X						
7. Is the response time requirement critical? (4.3)		X		X						
8. Are there host applications which will need this application's output? (4.4)		X		X						
9. Is there one other discipline which will use the application's output? (4.4)		X		X						
10. Are there five disciplines which will use the application's output? (4.4)		X		X						
11. Do all of the disciplines use the application's output? (4.4)		X		X						

Application	3.2.4		3.2.5							
Control Considerations	Y	N	Y	N	Y	N	Y	N	Y	N
1. Can the application run in the background? (5.1.2)		X		X						
2. Does the application have communication needs with applications on the host/mainframe? (5.2)		X		X						
3. Is the application independent of applications on the workstation? (5.2)	X		X							
4. Is the application dependent on applications on the host? (5.2)		X		X						
5. If there are communication needs, does the application have infrequent communication needs with applications on the host/mainframe? (5.2)		X		X						
6. If there are communication needs, does the application have periodic communication needs with applications on the host/mainframe? (5.2)		X		X						
7. If there are communication needs, does the application have heavy communication needs with applications on the host/mainframe? (5.2)		X		X						
8. Does the application have infrequent communication needs with more than one workstation? (5.2)	X		X							
9. Does the application have periodic communication needs with more than one workstation? (5.2)		X	X							
10. Does the application have heavy communication needs with more than one workstation? (5.2)		X	X							

Application	3.2.4		3.2.5							
Control Considerations (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
11. Is the execution time a critical factor? (5.2)		X		X						
12. Does the application require shared data access? (5.3)		X		X						
13. Does the application access the shared data frequently? (5.3)		X		X						
14. Does the application modify shared data? (5.3)		X		X						

Application	3.2.4		3.2.5							
Networking Delay Impact	Y	N	Y	N	Y	N	Y	N	Y	N
1. Are there more than eight message types required to be sent or received within this application? (6.4.2)										
2. Are there more than 32 message types required to be sent or received within this application? (6.4.2)										
3. Are the average message lengths greater than 100 bytes? (6.4.2)										
4. Are the average message lengths greater than 1000 bytes? (6.4.2)										
5. Are more than 10 messages per minute expected to be sent for this application? (6.4.2)										
6. Are more than 1000 messages per minute expected to be sent for this application? (6.4.2)										
7. Is the hardware network transmission speed less than 1 Megabit per second? (6.2)										
8. Is the hardware network transmission speed less than 10 Megabits per second? (6.2)										
9. Is the full protocol used to communicate at each level of the OSI model? (6.3)										
10. Is the BER of the network estimated to be worse than one bit in 100,000? (6.4.1)										
11. Is the BER of the network estimated to be worse than one bit in ten million? (6.4.1)										

Application	3.2.4		3.2.5							
Networking Delay Impact (Continued)	Y	N	Y	N	Y	N	Y	N	Y	N
12. Does this application program have data messages for more than eight user destinations? (6.4.2)										
13. Does this application program have data messages for more than 32 user destinations? (6.4.2)										
14. Does this application have more than eight users who can make data inquiries? (6.4.2)										
15. Does this application have more than 32 users who can make data inquiries? (6.4.2)										
16. Is the network a connectionless service? (6.4.2.2)										
17. Is the network utilization rate expected to be above 50 percent after adding this application? (6.4.2.2)										
18. Is the greatest peak communication demand of this program above 10 percent of the network transmission speed? (6.4.2.2)										