A number of performance prediction methods are available to IBM marketing personnel. This paper describes one such method, which predicts the effects of changes in IBM 3790 and 8100 distributed processing systems and in teleprocessing networks. Such changes may involve system features (such as line protocols), the introduction of new applications, or volume growth in an otherwise static system. The technique makes use of a detailed simulator, informally called FIVE, in conjunction with a system monitor and data analysis program. Its use can make substantial performance information available at relatively low cost.

The role of detailed simulation in capacity planning

by H. C. Nguyen, A. Ockene, R. Revell, and W. J. Skwish

From a capacity planning viewpoint, a system can be divided into three broad areas: host processor, communication network, and distributed intelligence, as in the IBM 3790 and 8100 systems.[1,2] Attention in the past has generally been focused on the host system, but the sharply declining cost of central hardware, combined with the rapid growth of teleprocessing and distributed processing, has increased the importance of communications and distributed intelligence.

Systems with dozens of distributed processors are no longer uncommon. Managing the growth in capacity of such systems requires close attention to key performance considerations. The basic requirement for capacity planning is predicting how changes will affect system performance, thereby permitting the best course of action to be followed. For purposes of the following discussion, the key performance elements are assumed to be response time (as perceived by a terminal operator) and throughput (volume of work per unit time, such as characters printed per second).

Methods for predicting system performance include "educated guesswork," benchmarking, and the use of computer models (either analytic or simulative, as discussed below). The first method

is time-honored and probably the most widely used. It works best when the system is comfortably overdesigned, allowing a safe margin for error. This casual approach to performance prediction has become increasingly unsatisfactory because of the growth of on-line systems (with their stringent performance requirements) and diminishing tolerance for gross overdesign.

Benchmarking (the execution of actual applications on the proposed system configuration) is expensive and inflexible, since it is difficult to examine a wide range of configuration alternatives. For a combination of flexibility (in specifying both system configuration and applications at any level of detail) and reasonable cost, computer models have come into widespread use. Such models are programs that describe both the physical structure of the system under study and all application activity (data transmission, processing, I/O) that significantly affects performance.

Once the decision has been made to employ a modeling approach, a number of factors influence the choice of a specific tool:

- It should be suitable for modeling individual subsystems (such as the 3790 and 8100) as well as the total system (communication network, host computer, and cluster controllers).
- The model should be detailed and accurate enough to simulate all relevant aspects of a working system, although the cost and level of detail must be kept in balance, and the model should minimize distortion. Errors caused by faulty input data cannot be avoided, but the model itself should not introduce significant error.
- There should be a reasonable validation procedure. Validity is established by matching the model to the operation of the real system, thereby providing credibility for the performance predicted when changes are made.
- The cost of using the model should be low to encourage its continued use. Aside from its value in predicting the effects of system changes or new applications, such a model can be used as a tuning aid, as in optimizing Network Control Program parameters.
- Model output should be provided at several levels of detail, as required by the nature of the study. Detail should not be forced on the user, but invoked as needed.
- There should be an automated procedure for using existing data. For example, a procedure for converting running application code to model input would save time. Similarly, tracing and monitoring facilities can provide performance data against which the model can be validated.

A key point is that information on both the causes of performance problems (such as poor data set placement) and the consequent effects (such as excessive disk utilization and concomitant

queuing delays) is usually available from existing system mapping and monitoring tools (to be discussed below). Unfortunately, those tools cannot answer *what if* or *how to improve* questions, a manifest need if the effects of possible remedial actions are to be predicted.

Why and *how* questions can be answered only if *cause* and *effect* data can be related, so that effects can be traced back to causes. They can be related by using the system model as an intermediary, feeding the *cause* data into the model and using the (measured) *effect* data to validate the model's performance estimates. The model discussed in this paper satisfies that requirement, relating measurable performance effects to observable causes.

The nature of modeling requirements, especially with regard to cost and timeliness, makes it impractical to construct a unique model for each installation. What is required is a generalized framework for modeling teleprocessing and distributed intelligence systems, with flexibility in specifying both configurations and applications. Such a framework exists in the modeling architecture discussed below. Used by IBM marketing personnel, it is informally called FIVE.

FIVE is a modular simulator of SNA (Systems Network Architecture) systems.[3] The simulator is a set of performance prediction aids within a common architecture, which is general enough to encompass other aids, such as configurators and network optimizers, although such other aids have not been implemented at this writing. The design allows subsets of the main simulator to be used interactively at IBM locations around the world.

Background

Analytic modeling for performance prediction[4,5] depends on solving a mathematical description of a system and its activity, an approach that has been most successful with individual subsystems. As such subsystems are linked together, and as the complexity of both configuration and applications increases, it becomes increasingly difficult to formulate (and solve) the mathematical relationships without making unrealistic simplifying assumptions. For example, no analytic model available today is capable of modeling a detailed Network Control Program (NCP) and predicting the effects of *slowdown* (whereby the NCP stops requesting new transactions when the number of free buffers drops to a critical value). In an Advanced Communication Function environment, where an overloaded line in one domain may force an NCP in another domain into slowdown, an analytic approach becomes even more futile.

The most general technique for the construction of performance prediction models is discrete-event simulation. Rather than relying on a mathematical description of a system, a simulator moves the system through time and mimics the myriad events that occur in the real world. Messages in a teleprocessing system are transactions that flow through the simulated system using necessary resources (terminal, communication line, CPU, channel). The simulator maintains a clock and keeps lists of transactions in various stages of processing and queuing for system resources.

Since a simulator is a detailed mapping of the real system into a computer program, rather than a mathematical abstraction, there are no inherent constraints on the level of detail that can be modeled. The limiting factors are the programming effort and the computer resource (storage and CPU time) available for executing the model. The rapidly declining cost of computer hardware has made detailed simulation a practical alternative in many cases, and it is that approach which was chosen in developing FIVE.

Once the decision was made to use a simulation approach, the next task was to select an appropriate modeling language. The most commonly used simulation language, GPSS,[6] was rejected on grounds of execution efficiency. After consideration of several others, the language finally selected was PL/I,[7] with some enhancements to provide the services required for simulation. This combination has the list processing facilities required for simulation and also produces efficient, executable code. PL/I has the further advantage of being a general programming language, so that it is possible to contemplate future (nonsimulation) design tools that use a data base in common with the performance predictor.

Architecture

The FIVE architecture can be viewed as a data base whose basic units are data structures called nodes. Each node contains the information required to describe one of the system elements (hardware and software) under study, and it can be used at different levels of detail. Intended to permit development of tools for all aspects of system design using a common data base, the architecture imposes a discipline that can speed development, validation, and future modification and maintenance by others.

The nodes in FIVE normally are linked in a hierarchical tree structure, although ring and mesh structures are also permitted. The hierarchy is: device (screen or printer) at the lowest level, then control unit, communication line, and transmission control unit (TCU), with the host processor (CPU) at the highest level. Local control units can be attached directly to the CPU. If the device is a disk or tape drive, the hierarchy is as follows: device, storage

control unit, channel, and CPU. Additional levels can be introduced by using such elements as a remote TCU or a cross-domain link.

In addition to hardware characteristics (such as storage size and processor speed), software characteristics appear in the nodes as appropriate. Using NCP generation parameters as an example, PACING parameters appear at the device (screen or printer) level, PASSLIM is in the control unit, PAUSE in the line, and SLODOWN in the TCU. A node can operate at any of several levels of detail; for example, the communication-line node can operate accurately either with all polling messages modeled or in a *fast* mode, with most unproductive polling suppressed to provide faster execution (although the term *fast* is inappropriate if the effect of limited TCU engine capacity is to be studied). Another example is the CPU node, which can be anything from a simple processing delay to a hierarchical task management facility.

In addition to the nodes, the FIVE input specification includes provision for defining logical files and their location (on disk or tape), characteristics of tasks in the CPU, format statements for controlling screen and printer I/O, and various probability distributions. Once the nodes have been linked into a network, a simulation facility models the dynamic behavior of the system. This facility provides such services as a clock, list handling, and event scheduling.

The nodes described above define the physical structure (and associated software characteristics) of the system to be modeled. The applications to be processed are defined by the Application Workload Description (AWD), a facility for describing the data transmission, processing, and I/O requirements for each type of transaction. Like the nodes, the AWD can operate on any of several levels of detail. The most basic form (type 0) is a single statement that gives information such as operator "think time," I/O characters transmitted, and processing time. A more complex version (type 1) permits the user to write a program with macros such as SEND, PROCESS, DELAY, and GOTO. In either case, all processing is in the CPU. To define an application in the IBM 3791 or 8100 controller, AWD type 3790 or 8100 permits the use of macros unique to the controller being modeled. There is also a *general control unit* feature which permits the user to define the characteristics of intelligent controllers other than the 3790 and 8100.

Distributed processing

Since distributed processing involves several intelligent controllers or processors within a single system, it is theoretically possible to apply the approach often used in capacity planning for

host systems; that is, to monitor subsystem resource usage levels and, knowing each subsystem's past and present resource usage, extrapolate its performance into the future. This approach has two major drawbacks, however. First, it can be misleading if the extrapolation involves changes other than increased load or a relatively straightforward system change such as additional disk drives. For example, if new applications or the introduction of shared disk storage are involved, extrapolation may be inaccurate. Second, regular evaluation of the performance of each subsystem can become a formidable task for all but the most trivial cases.

monitoring aids

Maximum value can be obtained from a performance model when it is used in conjunction with system mapping and monitoring aids. For the IBM 3790 and the 8100 operating under DPCX,[8,9] the following aids are potentially valuable:

- SYSDC—an *effect* aid, a software monitor that supplies data on the use of subsystem resources. Specifically, it reports control-unit utilization, disk utilization, and seek activity for a specified period. For our purposes, SYSDC provides an independent source of performance data.
- SYSLDSA—a *cause* aid, an analysis program that provides a complete map of data sets as allocated to the disk space of the 3790 or 8100. For our purpose, the information it supplies can have a major effect on performance and is inexpensive compared with manual data collection.
- FIVE3790 and FIVEDPCX—specialized subsets of the FIVE model. Each permits the use of macros similar to those used in actual 3790 and 8100 code. The controller is linked to other nodes in the system as described earlier with regard to FIVE. These two subsets of FIVE are available only outside the United States.

performance factors

The aids described above provide a ready source of cause and effect data, and FIVE can relate the two, but the information is still insufficient for construction of a model because the only *cause* information relates to data sets. The factors that affect performance of the 3790 and 8100 are those that determine resource availability and govern their use, namely:

- Configuration (hardware and software).
- Data-set activity (placement and usage).
- Application programs.
- Traffic volumes.

Information on subsystem configuration and traffic volumes should be readily available and need not be discussed further. Data-set placement information is supplied by SYSLDSA, as discussed above. Data-set usage, which is related to application program activity, is discussed below. Finally, information on exist-

Figure 1 Mapping of FIVE macros for source-program instructions

3790/8100(DPCX) Instruction		FIVE Macro Equivalent
RDDEV DISPLAY,TYPE=NOWAIT		RD FLDS=3,CHAR=20,OPT=NOWAIT
FREE DISPLAY,INPUT		FREE DPLY
MOVEBUF FROM=DISPLAY,TO=BUF(1)		MOVEBUF FROM=DPLY,TO=1
RESCON OPTION=WAIT		RESCON DPLY
DRL BUF(1)		DRLADD DSID=X,SYN=1.2,BUF=1
GETPANEL DISPLAY		GETPANEL SIZE=3
WRTDEV DISPLAY,END		WRT FLD=10,CHAR=400

ing application programs is readily available in the form of source code. That information is the most important, since application programs are the source of all resource usage. Whether the resource usage involves disk accesses, processor activity, or buffer usage, its control is determined by the flow of the application programs being executed.

The 3790 and the 8100 with DPCX are interpretive machines, and the execution flow of programs has a fundamental influence on performance. There is no great overhead of system code that needs to be executed. Thus, if the path lengths of all the application programs in a particular machine were doubled, one would expect the controller utilization to roughly double (assuming low utilization). In a traditional host system, on the other hand, the overhead of the system control program, access methods, and DB/DC packages has a considerable effect, to the extent that utilization attributable to application program execution itself may be relatively minor. Since this is not true for the 3790 or 8100, the effort required to map every source program into model input can be excessive.

Consider now that the FIVE model accepts as input a description of the application source programs in the form of a one-for-one mapping—one FIVE macro for each source program instruction, as shown in Figure 1. At first it may appear that the amount of work has increased. However, a potentially massive task can be reduced substantially by using a translation program which converts each real source instruction to its equivalent FIVE macro. Such a translator, known informally as SIX, has been written at the IBM United Kingdom Field Systems Centre at Croydon, England.

Two things soon become apparent. First, data gathering, a major task in any modeling effort, has been considerably reduced, so that it is practical to use the model in capacity planning. Second, the idea of automatically translating *cause* data into model input is not peculiar to the 3790 or 8100; it can also be applied to other system components. For example, all the *cause* data required to accurately reproduce an installed communications network in

FIVE is provided by a line trace, an NCP generation listing, and the Network Performance Analyzer (NPA), a network management aid, which is discussed under *Network capacity planning*, below.

The distributed processing model

Although it is feasible to gather a large portion of the required data quickly and with little effort, it would be misleading to say that the entire procedure can be automated. The following paragraphs describe the steps involved in representing a real system in the FIVE model and validating the model's predictions. Input to FIVE consists of three main components: a definition of the physical configuration, a definition of the applications and the transaction flow, and a file definition.

physical configuration

Mapping of the physical configuration into FIVE is not described here. It is a straightforward process, requiring only fundamental information on the system's physical characteristics, as described in the discussion of node structure under *Architecture*, above.

application program selection

For simplicity, the example presented below considers a system with one main program. It is based on a real situation, although most systems have many programs, often a hundred or more. It would require considerable effort to incorporate all programs into a system model; fortunately, only a few programs are exercised most of the time. Most programs relate to infrequent events, such as error conditions and weekly report generation, and can therefore be ignored on the assumption that their effect on system performance is negligible (although the option remains open to incorporate them if they are deemed important).

application program translation

The selected program is now translated using SIX. The result is a file that contains the required FIVE input statements cross-referenced with the original program statements, together with the locations of the program page boundaries. At this stage the SIX output needs manual modification before it can be used as FIVE input.

Almost all the operands required by the FIVE macros can be deduced by SIX, but a few must be supplied by the user. The information required is generally simple and easy to obtain, either directly from the source code or from SYSLDSA. Examples are whether the program accesses the transaction data set, the names of any external programs (subroutines) that are called, and the average number of synonym accesses to a particular indexed data set (supplied by SYSLDSA).

A more time-consuming step is coding the flows through the program. Although SIX can map the source code into equivalent FIVE

Figure 2 Unconditional branching—no manual intervention required

```
        Source Code          |          SIX-generated code
                             |
                             |          GOTOU*
          GOTO  XXX          |          GOTO  LABEL=LAB021,BLOCK=1
          ----- ----         |          -----   ----
          ----- ----         |          -----   ----
        page boundary        |          BLOCK
          ----- ----         |          -----   ----
          ----- ----         |          -----   ----
                .            |                .
                .            |                .
          ----- ----         |          -----   ----
  XXX     ----- ----         |  LAB021  -----   ----
          ----- ----         |          -----   ----
```

* This macro allows for execution overhead only

Figure 3 Conditional branching—probabilities to be user-supplied

```
        Source Code          |          SIX-generated code
                             |
                             |          GOTOC*
          GOTOC GE,XXX       |          GOTO  LABEL=MIX009,BLOCK=1
          ----- ----         |  LAB020  -----   ----
          ----- ----         |          -----   ----
        page boundary        |          BLOCK
          ----- ----         |          -----   ----
          ----- ----         |          -----   ----
                .            |                .
                .            |                .
          ----- ----         |          -----   ----
  XXX     ----- ----         |  LAB021  -----   ----
          ----- ----         |          -----   ----
                             |
                             |  MIX009  MIX   LAB020/%,LAB021/%
```

* This macro allows for execution overhead only

macros, it cannot determine the probability of taking a particular path at program branch points. That probability must be determined manually.

transaction flow definition

At every branch point in a program, the SIX translator produces two FIVE GOTO macros. The first accounts for the overhead associated with executing the branch instruction. The second causes the branch operation to occur; it has an operand, LABEL, which gives the branching address. As shown in Figures 2 and 3, if the branch is unconditional, SIX generates the operand value and no manual intervention is required; and if the branch is conditional, SIX generates a MIX statement which indicates the probability of the branch. This probability must be supplied by the user.

Determination of the MIX probabilities is best done in order of importance; that is, by successive selection of the most basic branch points, starting with those that branch to different func-

Figure 4 File definition for relative data sets

```
KEY:

aa      - data-set ID
nnnn    - data-set size in blocks
vv      - volume number
ttt ss  - track number and block number of first sector
rrrr    - number of records in the extent

 ┌──────────────────────────────────────────────────┐
 │                  S Y S L D S A                   │
 │        Relative Data Set information produced    │
 ├──────────────────────────────────────────────────┤
 │                                                  │
 │              RELATIVE DATA SET aa                │
 │                                                  │
 │              nnnn TOTAL BLOCKS                   │
 │                                                  │
 │              vv ttt ss nnnn rrrr                 │
 └──────────────────────┬───────────────────────────┘
                        │
                        │
                        │
                        │
                        V
 ┌──────────────────────┴───────────────────────────┐
 │                  F I V E                         │
 │       Relative Data Set information required     │
 ├──────────────────────────────────────────────────┤
 │                                                  │
 │       FILEaa  FILE TYPE=3791,                    │
 │               NREC=(rrrr,vv),                    │
 │               LRECL=256*(nnnn/rrrr)              │
 └──────────────────────────────────────────────────┘

* The position of the ttt ss values supplied by SYSLDSA
  is defined in FIVE according to the sequence in which
  files are defined. Both SYSLDSA and FIVE produce disk
  maps, which allow cross-checking to validate the file
  allocations.
```

tions within the program. These branch points will roughly equate to those shown in a block diagram. Next, it must be determined whether each function is to be exercised. Just as a good many programs are executed infrequently, many functions within a program are used rarely. Once these functions have been identified, it becomes a simple matter to code the related MIX statements so that they are not exercised (by setting the probability to zero). After the first-level MIX statements have been completed, the process is repeated for branch points within each function to be included in the simulation.

It is important that the MIX statements be coded accurately. The MIX probabilities are best determined by a person who is familiar with the application programs, usually the programmer himself.

file definition

At this stage the major part of the model building is complete. The next and final stage is to describe the data sets to be used. This operation is straightforward, since all the information required by FIVE is supplied by SYSLDSA, as shown in Figure 4.

model validation

Once the model has been completed, it can be validated by comparing it with available measurement data supplied by SYSDC. The

use of SYSDC requires only that it be initiated from a terminal attached to the 3790 or 8100 being measured. Once SYSDC is initiated, measurement data can be printed any number of times during the measurement period, which can be up to an hour.

Since there are several intelligent controllers in a typical system, it is impractical to attempt to measure them all. If a single system is used in all operating environments, normally it will be the one chosen for measurement.

The specific situations to be measured must now be selected. It is desirable to measure the system once for each application and also with each common application mix—for example, with eight terminals doing data entry and six doing inquiry. The environment in which measurements are to be taken must also be chosen: Are the measurements to be taken in a controlled environment with the terminals operating continually with fixed operator delays, or is the natural environment of the production system to be measured with no artificial constraints on its operation? Both approaches have merit and both should be used. In the former, operator delays can be controlled, so model validation cannot be affected by inaccurate choice of operator delays. A natural production environment, on the other hand, has the advantage of providing an overall validation of the model. In general, as many measurement runs as can be afforded should be undertaken, bearing in mind that the objective is to validate the model, not to use the measurement results alone for planning.

possible sources of error

If the measurement results do not compare closely with those predicted by FIVE, it is reasonable to conclude that there are errors in the model input data. Since most of the input is a direct translation of observed fact, there are relatively few potential sources of error. Among them are inaccurate operator delay values, incorrect program flows, and incorrect data set placement.

Operator delays have a fundamental effect on predicted results. If the system has not been measured in a controlled environment using fixed operator delays, such measurement should be considered so that operator delays can either be ruled out or highlighted as a source of error. If the delay estimates must be improved, we know of no better way to measure them than with a stopwatch.

To check the accuracy of program flows, FIVE incorporates two powerful facilities, SUMRY and TRACEX. SUMRY provides (among other things) a count of every instruction in each program executed during a FIVE run. It allows areas of overactivity and underactivity to be detected. Once suspect areas of the program are located, TRACEX can trace the program flow, showing which instructions are being executed, in what order, and how much time is spent on each.

Finally, data set placement can be checked by comparing the disk map of the real system, as supplied by SYSLDSA, with the disk map produced by FIVE.

revalidation

Since the production system almost certainly will change over time, it is wise to resynchronize the model with the real system from time to time. Such revalidation need not be as exhaustive as the initial validation, since only the changes need be considered. Revalidation should be considered when the system configuration changes, when new applications are added, and when there is a substantial growth in volume.

examples

A wide variety of situations can be investigated easily and quickly through the use of FIVE. A few examples follow:

Hardware and software—

- Of what value is the 3790 fixed-head feature, and which programs are best located under fixed heads?
- Is there value in keeping some programs in main storage? If so, which programs?
- What performance improvement can be expected by upgrading a 3790 system to an 8100 with DPCX? Where should 8130 controllers be installed and where are 8140 controllers needed?
- Will performance be enhanced by using direct sector referencing?
- How will performance be affected by driving some terminals over a remote data link?

Growth—

- When might expected volume growth produce unacceptable performance?
- What is the effect of adding a new application?

Performance analysis—

- Where are the bottlenecks in a system and why do they exist? What action is required to rectify the problem?
- What is the expected behavior of the system under various loads?

Almost all the above questions can be answered simply by changing a few FIVE parameters and rerunning the model. In this manner the use of FIVE, as opposed to the real system, allows experimentation. Alternative courses of action can be investigated without disturbing the production system.

other benefits

The approach described above is not restricted to capacity planning. The output produced by FIVE can also be used in system tuning. SUMRY and TRACEX, for example, are useful in examining the efficiency of program code, whether the program is used in production or in an application under development, and the data

set statistics produced by FIVE can help determine whether highly active data sets should be reloaded closer to the midpoints of the disks.

Network capacity planning

As with distributed processing, it is possible to build a FIVE model for capacity planning with a teleprocessing network. All required information exists in one form or another. First, it is necessary to determine whether sufficient *cause* and *effect* information is available to construct and validate the model, keeping in mind the need to maintain accuracy without undue effort. The sources of information, described below, are NCP generation, the Network Performance Analyzer, and line tracing.

NCP generation

A listing of the Network Control Program (NCP) generation is a valuable source of *cause* data for networks controlled by the IBM 3705 transmission control unit (TCU).[10] Many NCP generation macros significantly affect network performance, and much of the input required to construct a FIVE network model corresponds to NCP generation values.

Network Performance Analyzer

The Network Performance Analyzer (NPA),[11] a management aid for monitoring network activity, can play a vital role in the modeling approach to network capacity planning. It supplies both *cause* and *effect* data, thereby assisting in both building and validating the model. Relevant data supplied by NPA includes the polling rate (the number of polls per unit of time), 3705 utilization, and queue lengths in the 3705 (including the *cluster link outbound queue*, which contains data frames awaiting transmission from each 3705, and the *channel inbound queue*, which contains data awaiting transmission to the host). All of this data provides *effect* information of direct use in model validation, since it can be compared with similar output from FIVE.

The *cause* data produced by NPA includes buffer pool size (the number of buffers available in the 3705), and message rates by line, control unit, and logical unit.

line tracing

Normally used for investigating line problems, a line trace also contains information of value as model input. It provides details on the sequence of transmissions between end users in a network and, of particular value, it shows the messages generated by each host software user. For example, some systems may or may not issue an independent *unlock keyboard* message. While such information is available from other sources, it is often open to misinterpretation; the virtue of a line trace is the unambiguous nature of the information for the specific system and applications being modeled.

Much more information is available on communication between end users, such as the sequence of transmissions in a conversation, message lengths, and transmission frequency. Unfortunately, the format of the line trace output makes extraction of such information difficult. There is a need for a data reduction program to analyze line trace output. At present, familiarity with the applications must be relied on for the required input to FIVE.

The network model

The following example illustrates the data requirements of the FIVE network model and the level of detail incorporated into it. A single communication line is discussed for simplicity, but a network of arbitrary complexity (including multiple domains) can be defined in a similar manner.

Consider an SNA system that comprises two IBM 3274 (Model 1C) controllers, each with six 3278 (Model 2) display screens and a 3289 (Model 1) printer, attached to a single SDLC line. Three screens are dedicated to an inquiry application and three to data entry. The line is attached to a 3705 (Model 2) transmission control unit, which is attached to a host CPU. The FIVE code required to specify this configuration is given in Figure 5.

This is an adequate specification of the system configuration; default values are used for various unspecified parameters in each node. The left column in Figure 5 consists of user-defined labels, which generally are not necessary but are convenient for output identification. The DEVICE statements representing the screens specify the label of the Application Workload Description (AWD) that defines the data transmission, processing, and I/O requirements of each transaction type; also specified is the arrival rate (in transactions per hour) for each screen.

If control units with a different device configuration are attached to the line, another CU (control unit) statement should appear immediately after the last device in the example. A second line could be specified after all the control units and devices have been defined for the first line, followed by its CU and DEVICE statements. While not used in the example, the NUM parameter is also valid for defining the line; NUM>1 implies exact replication of the control unit and device configuration for each line.

Multiple transmission control units also can be defined. More than one CPU is permitted in an Advanced Communication Function configuration, and special parameters are available to specify attachment of a TCU to multiple CPUs, along with algorithms for transaction routing to applications in different host processors.

Figure 5 Code required for configuration specification

```
HOST   CPU
TC     TCU
LN     LINE
C      CU        TYPE=32741C,NUM=2
SINQ   DEVICE    TYPE=32782,NUM=3,AWD=INQ,ARRV=30
SDE    DEVICE    TYPE=32782,NUM=3,AWD=DE,ARRV=20
PRT    DEVICE    TYPE=32891
```

Figure 6 Additional parameters allowed for configuration specification

```
HOST   CPU      MODEL=4341
TC     TCU      TYPE=37052,STORAGE=32,BFRS=60,UNITSZ=100,MAXBFRU=10,
                  INBFRS=10,CHANTYPE=1,DELAY=2,BFRPAD=28,SLODOWN=12
LN     LINE     LNCTL=SDLC,SPEED=300,MODEM=(25,3),SDT=(CU01,CU02),
                  DUPLEX,CSD1=2,PAUSE=0
C      CU       TYPE=32741C,NUM=2,MILES=100,MODEM=(25,3),PASSLIM=8,
                  MAXIN=7,MAXOUT=7,MAXDATA=265
SINQ   DEVICE   TYPE=32782,NUM=3,AWD=INQ,ARRV=30,INRATE=3,
                  PACING=(1,1),VPACING=(3,1),APL,CRYPTO
SDE    DEVICE   TYPE=32782,NUM=3,AWD=DE,ARRV=20,...
PRT    DEVICE   TYPE=32891,LSIZE=80,PSIZE=66,BELT=94,PBUFFER=2048,
                  SCS,MAXRU=1700,PACING=(1,1),CRYPTO
```

Channels, disk and tape control units, and DASD and tape devices can be specified as additional nodes after specification of the CPU to which they are attached (before definition of the TCU and network). In this systematic manner, a host-network configuration of arbitrary complexity can be specified.

To illustrate more fully the input requirements (and the level of detail incorporated into FIVE), the nodes of the previous example are repeated in Figure 6, with some of the additional allowable parameters. The nodes are discussed briefly in the following paragraphs.

CPU node

A full discussion of the CPU node in FIVE is beyond the scope of this paper. Suffice it to say that, while FIVE contains no built-in models of access methods, system control programs, or DB/DC systems, there is provision for construction of detailed host models through the use of a hierarchical TASK structure coupled with detailed physical and logical file models. Such host representation must be coded by the user as part of the Application Workload Description (AWD).

TCU (3705) node

The 3705 and its Network Control Program (NCP) are modeled in substantial detail by FIVE. Except for hardware parameters (model number, channel adapter type, line scanner type), all the information required to define the 3705 can be abstracted directly from the NCP generation listing. The Network Performance Analyzer supplies data on the available buffer storage.

line node

Except for the service order table and PAUSE (both available from the NCP generation), the line parameters in Figure 6 are hardware oriented and should be readily available. SPEED is in characters per second, DUPLEX indicates data full-duplex operation, CSB1 is the TCU communication scanner base for this line, and MODEM gives the turnaround and transit times through the modem at the TCU end of the line (available from the manufacturer).

control unit node

The control unit parameters in Figure 6 are either hardware characteristics (MODEM is for the control unit end of the line) or available from the NCP generation. MILES is used to calculate the line propagation delay.

device (screen) node

In addition to the parameters already discussed, INRATE is the operator keying rate in characters per second (normally estimated from the nature of the application), PACING and VPACING are obtained from the NCP generation, and APL and CRYPTO are hardware features indicating APL and data encryption capability.

device (printer) node

With the exception of MAXRU and PACING, the parameters indicated in Figure 6 are all hardware oriented. MAXRU requires knowledge of the application, since it is a parameter of the BIND command used to establish a session with the printer. PACING is obtained from the NCP generation listing.

Application Workload Description

Figure 7 illustrates the AWD for a simple application on the system defined above. It is invoked by the screen node labeled SINQ in Figure 6. The transaction is started by sending a 30-character message from the screen to the host CPU (CHAR=30 in Figure 7). After 5000 instructions have been processed in the CPU, two output messages are sent: 1000 characters to the printer (DEST=PRT defines the destination as the printer label) and 200 characters back to the screen (the absence of DEST implies the screen at which the transaction originated). NEW indicates that the printer message is a newly created transaction, so that the original transaction remains in the CPU. Omission of this parameter would imply removal of the transaction from the CPU, which must not be done until the screen output has been sent.

Figure 7 Application Workload Description (AWD) for a simple application

```
INQ  AWD      TYPE=1
     SEND     CHAR=30
     PROCESS  INST=5000
     SEND     CHAR=1000,DEST=PRT,NEW
     SEND     CHAR=200
     AEND
```

The example given in Figure 7 is the simplest possible AWD for a "one-in, one-out" scenario (plus one message to the printer). Beyond this very basic scenario, FIVE enables a user to code with as much complexity as desired. To list several examples:

- A transaction can send chained (multiple) messages. It is also possible to code a scenario in which interaction is repeated between screen and host processor.
- A GOTO instruction can be used to code loops within an AWD or to exit to another AWD.

- A MIX (probability distribution) can be defined for such random variables as message length, processing time, and number of times through a loop. A mix of transaction types can also be specified at a screen by coding AWD=MIXLABEL rather than the label of a specific AWD.
- The PROCESS instruction can identify a required task; the task characteristics (such as priority and processing overhead) are defined in a separate TASK definition statement.
- Statements can be written to define the exact format in which information is to be printed or displayed; FIVE makes adjustments to allow for transmission of control characters and modifications of printer time.
- Assuming that the appropriate hardware nodes have been defined, disk and tape I/O operations can be interspersed with PROCESS macros.
- An intelligent control unit can be defined, so that processing can be done in the control unit in addition to (or instead of) the host CPU.

The AWD options, in conjunction with the flexibility in configuration definition and specification of hardware node details, provide a powerful tool for analyzing a network and planning for capacity expansion.

output

Several types of special output can be obtained before the normal output statistics, at the user's option. For example, the user may request a printout of the system configuration showing all nodes and their interconnections. If DASD devices are specified, a map showing how logical files are allocated to physical devices may be requested. Various forms of trace output can be produced, showing in great detail the operation of certain elements of the system (such as line, printer, and CPU). If the 3705 goes into *slowdown*, a trace of slowdown events will appear in the output.

After such special output, the first standard statistics are the AWD response times. Each AWD displays a number of observations (AWD completions), maximum, minimum, and mean response times, standard deviation of the response time distribution, and the time at the 90th percentile (90 percent of all responses will not exceed this value). A histogram of the actual distribution can be obtained by specifying HIST in the AWD header statement.

Following the AWD statistics, complete statistics are printed for each node, as listed in Table 1 (next page).

Concluding remarks

In summary, FIVE provides sufficient detail to permit capacity-planning decisions to be made quickly and accurately, and with-

Table 1 Following the AWD statistics, complete statistics are printed for each node in the network, as shown below

CPU—
- Overall processor utilization.
- Number of requests for each task.
- Utilization of each task.
- Percentage of time each task is seized.
- Queue statistics for each task.

TCU—
- Processor utilization.
- Number of times in slowdown mode.
- Percentage of time in slowdown mode.
- Free buffer pool information (average and minimum).
- Host queue statistics.
- Channel hold queue statistics (when in slowdown).

Line—
- Supervisory, inbound, and outbound utilization.
- Positive and negative poll utilization.
- Queue statistics, both in and out.

Control unit—
- Average poll cycle.
- Average waiting time for poll.
- Overall response time for screens on this control unit.
- Cluster link inbound queue (in TCU) statistics.
- Cluster link outbound queue (in TCU) statistics.
- Queue of messages waiting for a poll.

Device (screen)—
- Number of transactions generated.
- Maximum, minimum, and mean response time.
- Standard deviation of response time distribution.
- Transaction rate achieved.

Device (printer)—
- Number of messages processed.
- Average length of printer message.
- Number of lines and pages printed.
- Throughput in characters per second and lines per minute.
- Utilization.

Device (storage) and logical files—
- Utilization.
- Number of reads and writes per second.
- Maximum, minimum, and mean service time.
- Number of rotational-position-sensing misses.

out disrupting an ongoing operation. New applications can be accommodated, volume growth of existing applications can be planned for, and the effects of system changes can be assessed. The latter include such diverse elements as conversion from BSC to SDLC, screen and controller configuration on a line, installation of faster line scanners in the TCU, changing the printer mode from *data stream compatibility* to *SNA character string*, and conversion of disk I/O to fixed-block architecture.

The FIVE architecture and implementation facilitate the development of subsets for interactive use in IBM locations around the world. At this writing, two such subsets are in use, one for communications systems that use the 3270 family of controllers, and one for storage systems (tape and DASD).

validation

The need to validate a user-written model by comparing its results against available measurements was discussed earlier. A similar validation exercise was performed by the FIVE developers, using measurements made within IBM. In all cases studied, for both the network model and the detailed 3790 model, response times were found to agree within 10 percent; resource utilizations were even closer. Validation of FIVE is an ongoing activity, repeated as measurements become available on new products.

cost considerations

A note is in order on the cost in time of using FIVE in a capacity planning study. The following times typify the experience of one of the authors (Revell) at the IBM United Kingdom Field Systems Centre. All computer times given are actual CPU time on an IBM System/370 Model 168-3, and *days* are actually *man-days*.

For a typical 3790 study, obtaining the output of the SIX program required about two days; another eight days were needed for conversion to the final form needed by FIVE (assuming the availability of a person with intimate knowledge of the programs). Validation took another four days, using SYSDC output and stopwatch measurements of the actual system. Some 20 to 30 single-thread runs, each taking approximately 90 seconds of CPU time, were performed as part of this process. Next were about ten complete runs, each taking 15 to 20 minutes of CPU time; an additional four days should be allowed for these runs. Adding data collection and report preparation time, a total of six to eight man-weeks should be allowed for a study.

As a typical system, consider two 3705s, one controlling 20 lines, and the other 30. The 3705s are connected by a cross-domain link. Each of the 50 lines has two to five control units, with eight to 24 screens and at least one printer attached to each control unit. Approximately 60 single-line runs would be made during a system study, each taking about one minute of CPU time. Those runs would be followed by 10 to 15 complete network runs, each requiring 30 to 35 minutes of CPU time. Total personnel effort required for a network study of this magnitude would be somewhat less than for the 3790—perhaps five to six man-weeks.

ACKNOWLEDGMENTS

The authors wish to acknowledge the valuable contributions of Noel Sutton-Smith (IBM United Kingdom), the originator of SIX, and of Michael Gordon (IBM Poughkeepsie), Uwe Moysies (IBM Germany) and Jan Van Galen (IBM Netherlands), who were re-

sponsible for developing the 3790 and 8100 versions of FIVE. Peter Crank (formerly with IBM United Kingdom) was responsible for development of the interactive subsets currently in use on IBM's internal HONE system. And the necessary enhancement to PL/I was provided by Jerry Rubin, then at the IBM Scientific Center in Cambridge, Massachusetts, and currently at IBM's System Communications Division Laboratory in Raleigh, North Carolina.

CITED REFERENCES

1. *An Introduction to the IBM 3790 Communication System*, IBM Systems Library, order number GA27-2807, available through IBM branch offices.
2. *An Introduction to the IBM 8100 information system*, IBM Systems Library, order number GA27-2875, available through IBM branch offices.
3. J. P. Gray and T. B. McNeill, "SNA multiple-system networking," *IBM Systems Journal* **18,** No. 2, 263–297 (1979).
4. A. L. Anthony and H. K. Watson, "Techniques for developing analytic models," *IBM Systems Journal* **11,** No. 4, 316–328 (1972).
5. M. Reiser, "Interactive modeling of computer systems," *IBM Systems Journal* **15,** No. 4, 309–327 (1976).
6. G. Gordon, *The Application of GPSS V to Discrete System Simulation*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1975).
7. *OS PL/I Checkout and Optimizing Compilers—Language Reference Manual*, IBM Systems Library, order number GC33-0009, available through IBM branch offices.
8. *Operations Guide for the IBM 3790 Communications System, Version 7*, IBM Systems Library, order number GA27-2830, available through IBM branch offices.
9. *Distributed Processing Control Executive—Operations*, IBM Systems Library, order number SC27-0492, available through IBM branch offices.
10. *IBM 3704 and 3705 Communications Controllers Principles of Operation*, IBM Systems Library, order number GC30-3004, available through IBM branch offices.
11. *Network Performance Analyzer: Program Description and Operations Manual*, IBM Systems Library, order number SB21-2479, available through IBM branch offices.

GENERAL REFERENCES

T. E. Bell, "Objectives and problems in simulating computers," *AFIPS Conference Proceedings* **41**, Part I, 287–297 (1972).

E. K. Bowdon Sr., S. A. Mamrak, and F. R. Salz, "Simulation—A tool for performance evaluation in network computers," *AFIPS Conference Proceedings* **42,** 121–131 (1973).

R. W. Conway, "Some Tactical Problems in Digital Simulation," *Management Science* **11,** No. 1 (October 1973).

J. R. Emshoff and R. L. Sisson, *Design and Use of Computer Simulation Models*, The MacMillan Company, New York (1970).

P. H. Enslow Jr., "What is a 'Distributed' Data Processing System?," *Computer* **11,** No. 1, 13–21 (January 1978).

G. S. Fishman, *Principles of Discrete Event Simulation*, Wiley-Interscience Publishers, New York (1978).

S. R. Kimbleton, "The Role of Computer System Models in Performance Evaluation," *Communications of the ACM* **15,** No. 7, 586–590 (July 1972).

J. Reitman, *Computer Simulation Applications*, Wiley-Interscience Publishers, New York (1971).

M. F. Rothstein, *Guide to the Design of Real-Time Systems*, Wiley-Interscience Publishers, New York (1970).

P. H. Seaman, "On teleprocessing system design—Part VI, The role of digital simulation," *IBM Systems Journal* **5**, No. 3, 175–189 (1966).

A. C. Traub Jr. and W. F. Zachmann, "A GPSS Model of a Complex On-Line Computer System," *Proceedings, Symposium on the Simulation of Computer Systems*, ACM Special Interest Group on Simulation (June 1973), pp. 17–37.

H. C. Nguyen and W. J. Skwish are located at the IBM World Trade E/ME/A Corporation, Building 5, P.O. Box 390, Poughkeepsie, NY 12602; Arnold Ockene is at the IBM World Trade E/ME/A Corporation, 360 Hamilton Avenue, White Plains, NY 10601; Richard Revell is located at IBM United Kingdom Ltd., 17 Addiscombe Road, Croydon CR9 6HS, England.

Reprint Order No. G321-5117.