Practical problems in a distributed application

by ERIC D. CARLSON and MARY C. SMYLY

IBM Research Laboratory
San Jose, California

INTRODUCTION

In this report the term "distributed application" is used to refer to application programs with modules which operate on two or more computers, and which communicate with each other in real time. In particular, this report describes experience with an IBM Program Product, Trend Analysis/370 (TA/370). TA/370 is a distributed application employing two computers, a host and a satellite. The satellite computer is used for a color graphics feature, which can produce line graphs or bar charts in multiple colors. The host computer is used for data base management and non-graphical data transformations (such as computing ratios). Figure 1 shows the TA/370 system configuration, and details on the system can be found in the reference manuals.5,6 In TA/370, user queries are processed in the host computer. If the query contains a request for a graph or bar chart, the appropriate parameters (data and formatting) are transmitted from the host to the satellite. The satellite acts as a formatter and controller for the graphics terminal on which the graphs and bar charts are displayed. Thus, only the graphics formatting and control functions in TA/370 are distributed, and communication is from the host to the satellite.

Of the many possible reasons for a distributed application, the reason for distributing the graphics functions in TA/370 appears to be performance. Specialized graphics formatting functions, such as labelling and scaling, can be performed in the satellite based on encoded commands from the host. Thus, not all the formatting data for a graph need be transmitted from the host to the satellite (reducing transmission time) and the formatting commands can be executed in the satellite. Reduced transmission times and a special-purpose graphics formatter are intended to improve response time and reduce costs for creating graphical outputs in TA/370. To achieve these benefits, a special-purpose protocol for data communications was developed, and the satellite programs for graphics formatting were application-specific and coded in assembler language.

Our experience with TA/370 began with an attempt to modify the code for the satellite in order to be able to use different IO hardware than supported by TA/370. These were differences in implementation and not in functional capability. Because of the special-purpose nature of the satellite code and data communications protocol, we found modification difficult. Attempting to add new graphical functions to TA/370 appeared even more complicated. The difficulty in modifying and extending the graphical capabilities of TA/370 led us to investigate the possibility of using a general-purpose graphical subroutine package, developed by IBM Research for other applications,5 in TA/370. Because this subroutine package was developed to operate on the host computer, we expected that we would lose the performance advantages of the distributed processing for graphics in TA/370.

We implemented a prototype revision of TA/370 using the host-based graphics subroutine package. In comparing the prototype with the original TA/370, there was no noticeable change in response times. Further investigation of the two versions was made to find out why the theoretical performance advantage for distributed processing was not realized in practice. The results of this investigation indicated that the overhead in the host-satellite protocol was so large that advantages from reduced data transmission or parallel processing did not affect response time significantly. In addition, advantages from distributed processing appeared to be offset by need for data encoding and decoding to communicate between the host and the satellite. Use of a general-purpose software package operating on the host computer seemed to offer greater flexibility, easier maintenance and extensions, and reduced hardware and programming costs without any performance penalty compared to the original version of TA/370.

In the next section of the report we describe in more detail why we wanted to modify TA/370 and the problems we encountered in doing so. The third section describes the implementation of the "non-distributed" prototype and the benchmark comparison with the distributed TA/370. In the fourth section we analyze the results of the comparison, and in the fifth section we discuss some alternatives for distributed processing in applications such as TA/370.

THE PROBLEMS WITH THE DISTRIBUTED VERSION

We were interested in modifying TA/370 for two reasons: (1) to make it operational on a satellite computer with a different graphics terminal and host-satellite communications interface, and (2) to add new graphical functions (such
In making the modifications we encountered several problems:

1. The satellite code was special-purpose, designed for this application and satellite computer, and was written in assembler language. Even after six months of work, we never totally understood the satellite code, and the programmer who wrote the original code had to make the modifications. Even if we had been experts in the assembler language for the satellite, modifying the code would have been difficult because it required detailed knowledge of the hardware and of the data communications protocols.

2. Communication between the host and the satellite used a special-purpose set of records, with five basic record types, each of which was encoded at the bit level. That is, instead of using an extension of a standard communications format, such as the ASCII character set, TA/370 essentially used five special-purpose bit strings for which we had to learn the encoding and decoding.

3. The distribution of graphics function had resulted in duplication of functions in the host and the satellite computers. Specifically, the host modules scaled the data to be displayed into value ranges compatible with the range of values which could be represented in one word on the satellite, and the satellite rescaled the data into ranges compatible with the address space of the graphics display. In addition, the host modules encoded the graphics commands and data into the five special purpose records for transmission to the satellite, and the satellite modules decoded these records and then recoded the data for transmission to the graphics terminal. Thus to make any modifications to the graphics modules, one had to understand the details of both the host and the satellite coding schemes.

In looking at more complicated modifications of TA/370, such as adding new graphics functions or supporting a different type of graphics terminal, it was apparent to us that problems, such as those listed above, would become more severe. We would have to add and modify code in both the host and the satellite and we would have to modify or extend the data communications records.

EXPERIENCE WITH A NONDISTRIBUTED VERSION

Graphics functions in the nondistributed version

We had several years experience with an application similar to TA/370 in which the graphics functions were not distributed. This system used a general-purpose graphics subroutine package, called DISPLIB, which was developed to support a variety of graphics functions and graphics terminals. This package, written in FORTRAN, provides basic commands for displaying text, lines, and points, as well as for display transformation (e.g., scaling) and user input. The package is similar in style to the standard subroutine package currently being proposed by ACM SIGGRAPH. The package also uses extensions of the ASCII or Correspondence character sets for graphics communication to the terminal; this convention is used by several terminal manufacturers. Because the package operated on the host computer, its use in TA/370 required that we not distribute the graphics functions. We expected that this change to a nondistributed application might degrade performance.

A prototype version of TA/370 using DISPLIB was implemented. The prototype supported only simple bar charts and therefore was not functionally equivalent to the original TA/370. In addition, the prototype simply translated the five records (that TA/370 created to send to the satellite) into DISPLIB calls. It would have been more efficient to replace the TA/370 code that creates the five records with code calling DISPLIB, but simply translating the records required fewer changes to TA/370 and was easier to program.

Comparison: distributed and nondistributed

Table I gives a comparison of the graphics functions in TA/370 and those in our prototype revision of TA/370. This comparison indicates the basic differences between the two versions.

The prototype was implemented and a benchmark bar chart containing data on six organizations for five years
TABLE I.—Comparison of Graphics in TA/370 and TA/370 Prototype

<table>
<thead>
<tr>
<th>Feature</th>
<th>TA/370</th>
<th>TA/370 Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>graphics package language</td>
<td>assembler</td>
<td>FORTRAN</td>
</tr>
<tr>
<td>style</td>
<td>special-purpose</td>
<td></td>
</tr>
<tr>
<td>graphics function implemented on</td>
<td>satellite</td>
<td>host</td>
</tr>
<tr>
<td>graphics functions invoked by</td>
<td>building</td>
<td>subroutine calls</td>
</tr>
<tr>
<td>host code language</td>
<td>PL/1</td>
<td>PL/1</td>
</tr>
<tr>
<td>type of graphics</td>
<td>line graphs and bar charts</td>
<td>simple bar charts</td>
</tr>
<tr>
<td>communications</td>
<td>5 special record types</td>
<td>ASCII extension</td>
</tr>
<tr>
<td>what communicated</td>
<td>data + graphics</td>
<td>data + graphics</td>
</tr>
<tr>
<td>terminals per satellite</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>terminal types supported</td>
<td>1 (raster)</td>
<td>3 (raster, vector, storage)</td>
</tr>
<tr>
<td>size of host code for graphics</td>
<td>29K bytes</td>
<td>113K bytes</td>
</tr>
<tr>
<td>(approximate)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lines of host code written</td>
<td>500</td>
<td>170</td>
</tr>
<tr>
<td>by applications programmer (approx.)</td>
<td>47K bytes</td>
<td>24K bytes</td>
</tr>
<tr>
<td>satellite code size (approximate)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A (=30 bars) was used to compare performance. Amount of code and communication load also were compared, and flexibility and programming cost were compared subjectively.

1. **Response time:** For each version the time to draw the bar chart was about 15 seconds. (Note: in other installations of TA/370 which use a S/7 with hardware features to assist in host-satellite communications, response times for bar charts similar to the one used in our benchmark are about 10 seconds.)

2. **Communication load:** In the original TA/370 implementation about 800 bytes of bar chart values and formatting keys were transmitted from the host to the satellite. In the prototype version of TA/370 about 2500 bytes of bar chart values and formatting data had to be transmitted. Transmission was at 1200 Baud. TA/370 sends 268 byte blocks and the prototype used 256 byte blocks.

3. **Communications protocol:** Both versions use 1200 Baud asynchronous communication over voice grade lines. The original version uses a transmission code (Paper Tape Transmission Code) developed for IBM 2740-1 terminals. The prototype based on DISPLIB uses ASCII or Correspondence codes. The protocol in the original version uses five special-purpose data communications records with host-satellite “handshaking” after each block is transmitted. The prototype uses “standard” ASCII codes with “handshaking” only at the start of host-satellite communications.

4. **Amount of code:** Even though we implemented only a subset of the TA/370 graphics functions, the prototype version of TA/370 required much more code in the host because a general-purpose package was used. However, the applications programmer wrote less code in the prototype because this package was available. The code written by the applications programmer in the prototype is about 10 percent of the total amount of code, whereas in the original TA/370 it is 100 percent. We estimate that if we had chosen to implement only the TA/370 graphics functions (eliminating any DISPLIB code not required for these functions) and to rewrite the TA/370 host code for (creating graphics records rather than adding code to translate those records into DISPLIB calls), the amount of host code in both versions would have been approximately equal. Any implementation of the prototype would require less code in the satellite. Moreover, the satellite code used in the prototype would support all the TA/370 graphics functions and would not require modifications for most additional graphics functions.

5. **Size of satellite computer:** The original TA/370 required an IBM System/7 with disk storage, while the prototype could operate using an IBM 5100. The System/7 is about five times more expensive than the 5100 (about $50,000 more).

6. **Programming cost and skill level:** Because the prototype used a high-level language of graphics calls rather than having to create records at the bit level, we believe that the cost to program the prototype was less than the cost to program the equivalent function in the original TA/370. The prototype was developed in less than two person months for a programmer who was not familiar with the DISPLIB package. In addition, both the host and satellite code for graphics in the prototype would not need to be changed if new graphics functions were required. This type of general-purpose graphics...
would greatly reduce the programming cost for modifications of TA/370.

7. Flexibility: Because the prototype used a high-level language for graphics functions, and a general-purpose graphics subroutine package which supported several types of terminals, we believe that the prototype version was more flexible and would be easier to modify or extend than the original version of TA/370.

In summary, the experience with the prototype indicated that a nondistributed version of TA/370, using a general-purpose graphics subroutine package would not degrade performance, even though the communication load was increased, and would offer significant advantages in terms of hardware costs, programming costs, and flexibility. The performance results of the benchmark were not as expected, so the prototype and original versions were further analyzed to try to better understand the performance implications of the distributed versus the nondistributed implementations.

ANALYSIS: DISTRIBUTED VERSUS NONDISTRIBUTED VERSIONS

One could argue that the prototype performed as well as the original version of TA/370 because we implemented only a subset of the graphics functions of TA/370 and because we had the advantage having the original version as a model for implementing the prototype. Both these arguments are partially true, but further analysis indicated that even with equal functions and equal skill in implementation one should not expect much difference in performance between the distributed and nondistributed versions. The analysis indicated that the predominate factor in performance was the time to transmit data from the host to the satellite over the 1200 Baud line. For the bar chart used in the benchmark comparison the original version of TA/370 transmitted less (about one-third as much) data than the prototype version. However, in the original version of TA/370 there is "handshaking" between the host and the satellite once for every transmitted block. This "handshaking" is implemented in software. The "handshaking" consumes enough time to eliminate any performance advantage for the original TA/370 due to reduced data transmission.

This informal analysis of the performance of each version is substantiated by an analysis using a simple model. This model is a simplification of a model of distributed (graphics) systems proposed by Foley.\textsuperscript{3} The model consists of four factors which affect performance: data base access time in the host, execution time in the host, execution time in the satellite, and data transmission time. Thus for any graphics function in either version, a model of performance is:

\[
T = TD*ND + TH*IH + TS*IS + TT*TL
\]

where:

- \(T\) = total time (e.g., response time)
- \(TD\) = average time for data access in host
- \(ND\) = number of data base accesses
- \(TH\) = execution speed of host
- \(IH\) = instructions executed in host
- \(IS\) = instructions executed in satellite
- \(TT\) = transmission speed
- \(TL\) = transmission load

For either the distributed or nondistributed versions of TA/370, the \(TT*TL\) term in the model is the predominate factor. Transmission speed is 1200 Baud and the transmission load is in thousands of bits, so that \(TT*TL\) results in times measured in seconds. The data base access times (TD*ND) in both versions are equal because the same data must be retrieved. These times are usually in hundredths of a second. The host and satellite execution speeds are approximately equal because the satellite is about one-third as fast as the host, but the host is time-shared and time-sharing, on the average, reduces the effective execution speed by about one-third. The execution plus data base access times (TD*ND+TH*IH+TS*IS) in both versions of the system are in tenths of seconds. Thus, the model indicates that any performance difference between the two versions is going to be due to differences in transmission times. Detailed expressions for computing the transmission loads in each version were worked out, but are omitted here. In general, for a simple graph or bar chart the transmission load in the original TA/370 is about half that in the prototype (approximately 600 bytes compared to 1200 bytes). For a complicated graph it may be only one-sixth as much (about 1000 bytes compared to 6000 bytes). Note that at 1200 Baud the transmission load in the prototype version could take up to 40 seconds to transmit. In practice, however, the performance advantage that this simple model predicts for the original version is not realized. As previously mentioned, the practical failure to realize this advantage seems to be due to the overhead of the transmission protocol used in the original version.

ALTERNATIVES FOR DISTRIBUTED FUNCTION

Models of distributed applications

To choose among alternative configurations for a distributed application such as TA/370, one could use the model proposed by Foley.\textsuperscript{3} Given a set of alternatives for host, satellite, and data link hardware this model can be used to select the configuration which will produce the best response time for a given cost constraint. However, one needs to have predetermined the distribution of function and to have estimated the frequencies of graphic macro instructions, the probabilities of various user interactions, and the data base accesses required per interaction.

It would be very difficult to estimate the parameters of TA/370 and the parameters of hardware alternatives needed for the Foley model. More significantly, the hardware alternatives for TA/370 are restricted by the application's requirement for a color graphics terminal which could be attached to a host over voice grade lines. Therefore, the Foley
model does not provide much help in selecting among alternative distributions of function for TA/370. However, Foley used his model to analyze four "typical" graphics applications, one of which, two dimensional drawing, is similar to TA/370. Foley's analysis considered over 12,000 possible hardware configurations. Based on this analysis, Foley recommended that, in general, the best system configuration for graphics applications consisted of the simplest possible satellite (which provided enough function) using a voice-grade (2400 Baud) data link. Foley's analysis indicated that the data link and bulk storage on the satellite are the two components which most influence response time. Foley's analysis of distributed applications thus is very similar to our study of TA/370: data transmission load is the key variable affecting cost/performance in a distributed application, and distribution of function (or data) is useful only to the extent that it helps reduce this load.

Foley's model gives us insight in selecting a hardware configuration, given a set of functional modules for a distributed application. For a given hardware configuration and function modularization, a graph partitioning model, such as that suggested by Hamlin, can be used to decide which modules should be distributed. In a graph model of a program, the modules are nodes and among the nodes there are directed arcs which represent the calling and returning structure of the program. Associated with each arc is a cost, such as the amount of data passed or the overhead time for calling. Other costs, such as the amount of global data accessed from each module or the execution time for each module on the host and on the satellite, may be associated with each node. The graph partitioning algorithm attempts to minimize the cost associated with assigning the modules (nodes) to the host or the satellite. To do this, the model requires an estimate of the frequencies of data accesses and inter-module calls. For an example of such a model see Hamlin. As indicated by Hamlin's example, unless there are constraints (such as limited memory space) which prevent all modules from being assigned to one computer, the cost function usually is minimized if there is no distribution of function.

Note that both the Foley and Hamlin models assume that the program modules are defined. Yet the definition of the modules is a primary factor affecting the cost functions in both models. This suggests the need for a model based on the costs of performing various graphics functions, independent of program modularization, which could be used to analyze alternative modularizations. In addition, both models require many parameters of the application, program, and hardware. Estimating these parameters is difficult, and likely to be error prone. To the extent that the models are sensitive to errors in the parameters one would expect the models to suggest suboptimal alternatives which, if used, would need performance tuning. For performance tuning, it would be useful to be able to move program modules between the host and satellite computers. To make this easier, one needs to be able to compile and execute the same language on both machines, and to have a mechanism which supports calls and parameter passing among modules on different machines.

Possible alternatives for TA/370

The general guidelines of the Foley and Hamlin models for distributed applications suggest two alternatives for TA/370.

1. Reduce the size of the satellite computer and the modules assigned to it to as small as possible to contain the bar chart and graph formatting functions. Place all remaining modules in the host. This distribution will limit data transmission to data values only. The double encoding/decoding and data scaling currently in TA/370 will be eliminated. Our prototype is representative of this alternative.

2. Distribute the entire application program to the satellite, with batch transmission of a subset of the data base to each satellite on an as-needed basis. However, both the data base and the application program are too large for most satellites. A variation of this alternative would be to keep a working-set of the data base in the satellite (e.g., the data used in the previous "n" displays), and distribute all the graphics functions to the satellite, as done in the original TA/370. This alternative will reduce the data transmission because all the data for a graph or chart will not need to be transmitted if it is a variation on one of the charts stored in the working-set.

For either alternative it would be beneficial to use the highest speed communication link compatible with the applications' requirement for a voice-grade line. Currently 4800 Baud is possible, although to use 4800 Baud would require changes to the satellite hardware (System 7 or 5100) and to the communications protocol (bisynchronous instead of asynchronous).

CONCLUSIONS

The experience with TA/370 identified several practical problems in distributed applications:

1. The potential performance advantage of distributed function was reduced because of the overhead of host-satellite communication protocol.
2. Distributed function resulted in special-purpose programming and data encodings which made modifications and extensions difficult.
3. Distributed function greatly increased the cost of the satellite computer, without compensating savings in the cost of the host computer.
4. Distributed function probably increased the amount of code and the programming cost for this application.
5. Transmission time between the host and the satellite was the primary performance factor. The application's need for a 1200 Baud line limited the performance improvement that could be achieved by distributed function as long as the data base was stored on the host.
6. Deciding how to distribute function involves considering hardware configuration, alternative distribution of program modules and data, and alternative modularizations. Existing analytic models of distributed applications do not encompass all of these considerations.

7. In designing distributed applications such as TA/370 the constraints are: the available hardware, the hardware costs, and the software development and maintenance costs. As with most programs, the software development and maintenance costs are likely to be the most important constraint.

This study does not indicate that distributed processing is ineffective for applications such as TA/370. It does indicate that distributed processing is not simply a separation of function among computers, because the separation requires communication. Because communication is involved, design decisions can become more complicated (as indicated by Foley's model), and unexpected inefficiencies can result (such as the "handshaking" problem). Most importantly, when the design objectives change (e.g., added function or different hardware), a distributed implementation may be more difficult to modify than a non-distributed version. Our experience with TA/370 indicates that these "practical" problems can eliminate a theoretical advantage for a distributed application.

REFERENCES