PERFORMANCE ENGINEERING/MANAGEMENT OF A LARGE REXX APPLICATION

PAT MEEHAN AND PAUL HEANEY
IBM

Performance Engineering/Management of a Large REXX application.

Pat Meehan, Paul Heaney

ECFORMS[1] Development Team
IBM HSL Programming Systems Laboratory and Delphi Software Limited
Dublin, Ireland

Abstract

This paper addresses the performance engineering/management of a large REXX product (100 Kloc). This was in response to a market driven requirement to improve product responsiveness. The Software Performance Engineering methodology in conjunction with our own developed processes were used throughout. A number of Software Performance Engineering processes were adopted and became the basis for the drive towards performance improvement. A benchmark was developed with customer input. Targets for representative transactions were defined for our three main metrics, end-user response time, virtual CPU time and start I/O. Various REXX performance ideas/myths were validated/rejected on the basis of measurements. Measurement and analysis tools written in REXX were used to automate and assist in the continuous tracking of the targeted performance improvements. Bottle-necks in the code were identified by tracing and measuring some/all of each target transaction. The main performance engineering principles that were used were Fixing Point, Parallel Processing, Centering, Processing versus Frequency and Instrumenting. Follow up meetings with the IBM VM Laboratory's performance team in Endicott, discussions with Mike Cowlishaw and consultation with other REXX development sites has resulted in a pool of knowledge being built up on Performance methodologies and REXX performance rules/guidelines. Education of the development team resulted in REXX performance rules becoming instilled in the day to day design/coding of product changes. The product has a separate performance development team and eighteen months down the line focusing on performance has resulted in major improvements. Throughput in the two key service machines has been doubled and end-user response time has been reduced by a quarter. Further improvements have been prototyped which indicate we will be able to improve the throughput again on the service machines and significantly reduce the end-user response time.

Author Information

Pat Meehan B.E., M.Eng.Sc. joined IBM Ireland in 1984. He spent two years on assignment in IBM Netherlands where he worked on the performance of ECVM (EMEA Common VM) REXX applications. He then spent two years working on the design and implementation of a data extractor generator on MVS. Subsequent work included working on the future strategy of a program product, SAA/DM. In the last eighteen months, he has been responsible for the performance work being carried out on the REXX product offering, ECFORMS together with working as a consultant on performance to the rest of the development group.

Paul Heaney works with Delphi Software and is a REXX development consultant to IBM HSL Laboratory. He has been closely involved with the ECFORMS Product Offering over the past two years. He has concentrated his effort on the application of performance engineering techniques to the application over the past eighteen months . As a key member of the performance team, he also acts as a consultant to the rest of the

[1] ECFORMS is an electronic forms management system which is an IBM product offering and is a trademark of IBM.

development team. He has developed tools in REXX to assist in the measurement and analysis of the development effort.

Performance Engineering/Management of a Large REXX Application

Electronic Forms Management System (ECFORMS) is an IBM licenced program (5785-MCB) extensively used within IBM by many diverse applications and marketed in the US, Europe and Japan. It has three main functions, Forms Processing, Forms Design and Forms Administration running on the VM operating system.

The major features of ECFORMS Forms Processing are :

1. Filling in online forms (Origination).
2. Routing forms electronically.
3. Using electronic signature to :
   - Approve and Final Approve a form.
   - Reject a form.
   - Cancel a form.

Some of the main ECFORMS transactions (e.g. Filling in a form) involves communications via IUCV with two service machines in a serial fashion. There would be data validation by a data service machine followed by control checking and routing by an authorization service machine. In a multi-node situation, these service machines communicate with their peers through the use of spool files.

All of the three major functional areas are written, almost exclusively, in the REXX language.

This paper focuses on the performance engineering effort of the ECFORMS Forms Processing function.

Background

The market driven requirement to improve performance was identified by direct communication with our customers. The main performance issues were:

1. Response times from an end-user's point of view.
2. Throughput on the two key service machines.
3. CPU consumption and excessive I/O demands

These issues became the driving force behind the performance effort.

Approach

Most performance methodologies advocate (correctly) the application of performance engineering to systems in their early developmental stages. They concentrate largely on new systems and not on existing systems.

An example of such a methodology is that of Software Performance Engineering [1] (SPE). Closer examination of the methods encapsulated in this methodology highlighted a considerable degree of applicability to existing systems also. For this reason, SPE together with our own methods formed the basis of our approach to the performance effort.

Effort Allocation

The resource to improve the performance of the product has been spread over two different efforts. The initial effort (Effort 1) used three different development locations. Our own laboratory was one of these locations and also acted as the focal point for the other 2. The subsequent effort (Effort 2), currently ongoing, was concentrated in our own development laboratory and involved the setting up of a performance team for the group.

This paper focuses to a large extent on the performance work carried out in our own laboratory over the two efforts, although it draws on significant experiences from working with the other two locations.

SPE Methods

SPE is a methodology which advocates the application of performance analysis in the development of software systems. It provides a sensible method for the production of software that will meet certain

performance objectives. SPE encompasses the following methods [2]:

1. Design Principles which are an abstraction of the expert knowledge of performance specialists
2. Data collection is the means of acquiring the data necessary to describe the performance specifications
3. Modeling techniques uses execution graphing and analysis algorithms to predict performance
4. Proposal Evaluation
5. Instrumentation of the software system.

Several examples can be found in the literature of successful SPE usage [3]

SPE in its entirety was not appropriate for an existing product; we used those aspects of the SPE methodology which we found suitable for the task.

Adopted Benchmark

SPE recommends the use of a benchmark as an experiment for collecting performance data. The benchmark should be a good representation of the way the software is used by customers and should be easily reproducible.

The first step was to develop a benchmark to gather pertinent performance metrics. Compromises between representativeness and reproducibility had to be made due to resource and time constraints.

The main feature of the benchmark [4] were as follows:

- A set of transactions that were representative of the typical user workload scenarios based on a considerable depth of knowledge within the laboratory's support and development groups together with feedback from a subset of customers. A typical transaction would be to approve an ECFORMS form sent by an originator or for a service machine to process that approval request.
- A representative ECFORMS form
- Software operating conditions like compiled service machines at given priorities running on selected hardware. Due to resource constraints, the initial hardware was a 4381 running VM/SP5 but this has since been extended to include VM/XA on a 3083 and VM/ESA on a 3090.
- Single user on a single processor. This has been extended recently to include two processors communicating with each other for a subset of transactions.

The benchmark is an evolving experiment undergoing change as the software undergoes modifications and the user workloads shift.

Application of the SPE principles

The SPE Design Principles are a formalization of the performance knowledge of experienced performance engineers.

The principles of SPE were intended primarily for software creation, but we have found some of them to be equally applicable to a project which has undergone significant development work. However, it is conceded that the application of the principles is a more painful exercise at the later stages of a product's evolution.

The design principles have since formed the basis of our performance guidelines which we provide to our own and other development groups in the lab.

We now describe those adopted principles that were particularly applicable to the existing system and examples of that applicability.

1. **The Fixing-Point principle states that the connection between the data and the required result should be established as early as possible in the processing provided that the cost of retaining that connection can be justified.**

   - The product uses flat files as its file system. We found several cases where the product was accessing the same control files, several times in the same transaction sometimes for the same information.

     According to the principle, it made more performance sense to read selected files or sections of files into storage at initialization. Data was stored in a REXX array of the form x.y, where y was the key. Subsequent retrieval of information was then done from storage with great efficiency.

     Application of this principle was more appropriate to the service machines where initialization time was not a concern.

However, one of the constraints was that both service machines should still be able to run with 3meg of memory.

For some large files, the cost of retaining the connection for the entire file could not be justified because of the storage constraints.

The largest file used by the service machines is the organizational directory typically of the order of thousands of records.

Client information is retrieved by one of the service machines from this directory for auditing purposes. Here, we have prototyped the concept of a time window, where participating client information is extracted once from the directory within the time window and held in storage. Subsequent attempts to retrieve the same client's information, during the time window, is then done from storage.

In this way we can allow the installation control the cost of holding the information in storage by changing the time window.

2. **The Processing versus Frequency Tradeoff principle, says that we should consider the processing time of a transaction and the number of times that transaction is executed and minimize the product**

   - This principle can be satisfied in a number of ways. One of the less obvious ways is to expand the processing within a particular transaction to include another transaction ensuring that the new transaction is faster than the sum of the two old transactions.

     This can often be achieved by making maximum use of the overhead involved.

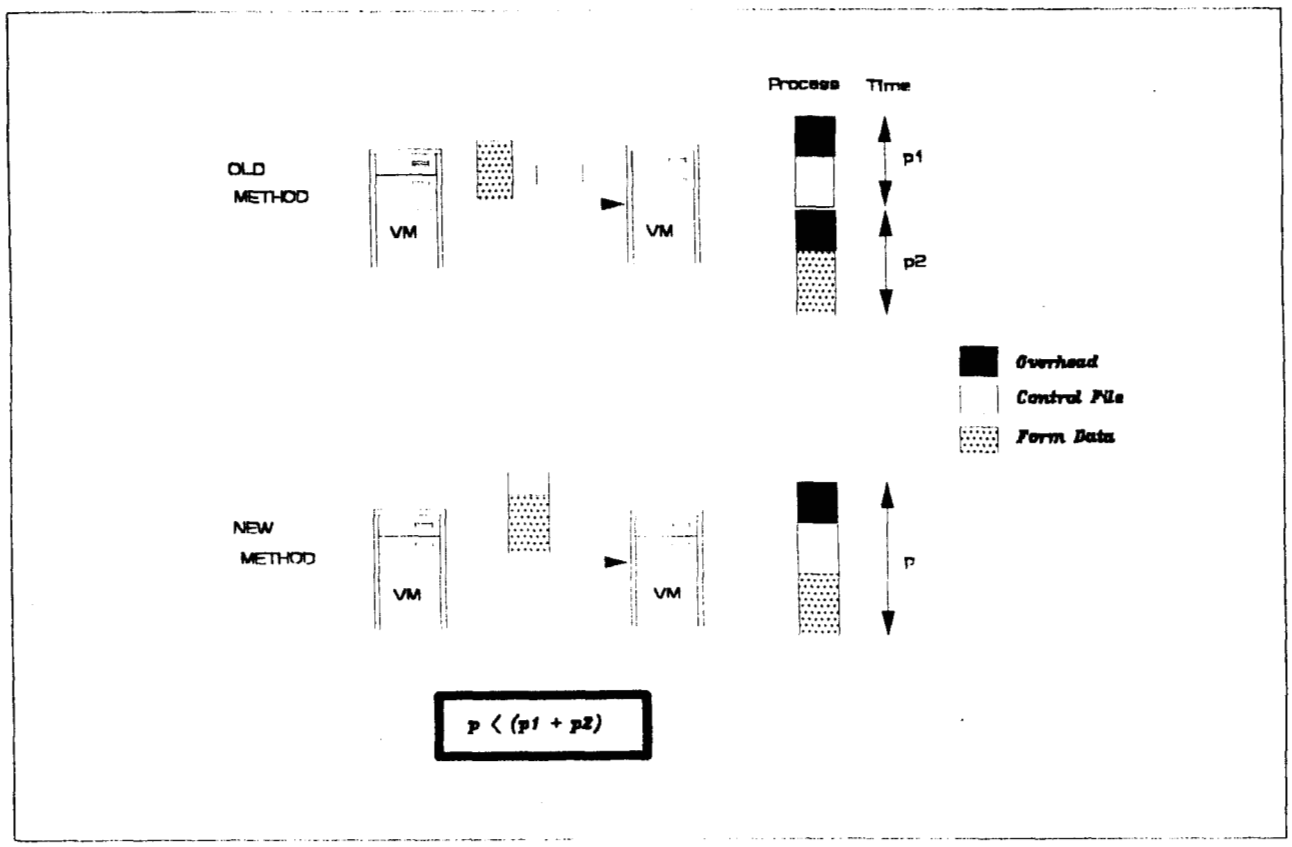

Figure 1. Processing-v Frequency Tradeoff Principle

The diagram (please refer to Figure 1) shows a typical application of this principle. Currently, when data is transmitted to participating nodes, a control file is sent and is followed by the form data file. The form data file has header information which associates it logically with the control file.

This means that the receiving node has the duplicate overhead of processing two spool files for each form data file and of estab-

lishing a logical association between the files received.

A change has been prototyped to make maximum use of this overhead, where the form data file is chained directly to the control file and a single file transmitted. The receiving server has only to process the one spool file thus ensuring that the product of the processing time multiplied by the frequency is reduced significantly.

Changes of this nature present significant architectural and migrational difficulties and underlines some of the headaches of performance engineering of developed systems.

3. **The Centering principle advocates the identification of the dominant workload functions and the minimization of their processing**

   - The transactions of form origination, approval and final approval were clearly the dominant workload functions and a large proportion of the effort was focused on these transactions.
   - A further application of the centering principle is of course within each of the dominant transactions. This performance refinement identified the dominant processes within the dominant transactions.

     Validation of form data is a typical example of a dominant sub-process. The form data is defined as a set of fields and associated values. Validation of form data occurs for all three dominant workload functions.

     Previously, during approval and final approval of each form instance, this validation process was again applied to the entire form data. A significant processing reduction was achieved on two dominant transactions by applying the validation process to the changed form data only.
   - As an indirect extension to the last point, it was realised from customer contact that form data approval and final approval often occurred without any changes to the data. When this occurred and was detected a large proportion of redundant processing was bypassed.

     Even more substantial improvements have been prototyped for this change in the multi-node scenario. Currently, the data is transmitted to participating nodes even if it is unchanged and the receiving nodes have to go through a lot of unnecessary processing to handle the large amount of spool files.

     A change has been prototyped where the form data is only transmitted when it is changed resulting in significant savings to the handling of remote requests by the service machine as illustrated by the diagram, Figure 2 on page ix.

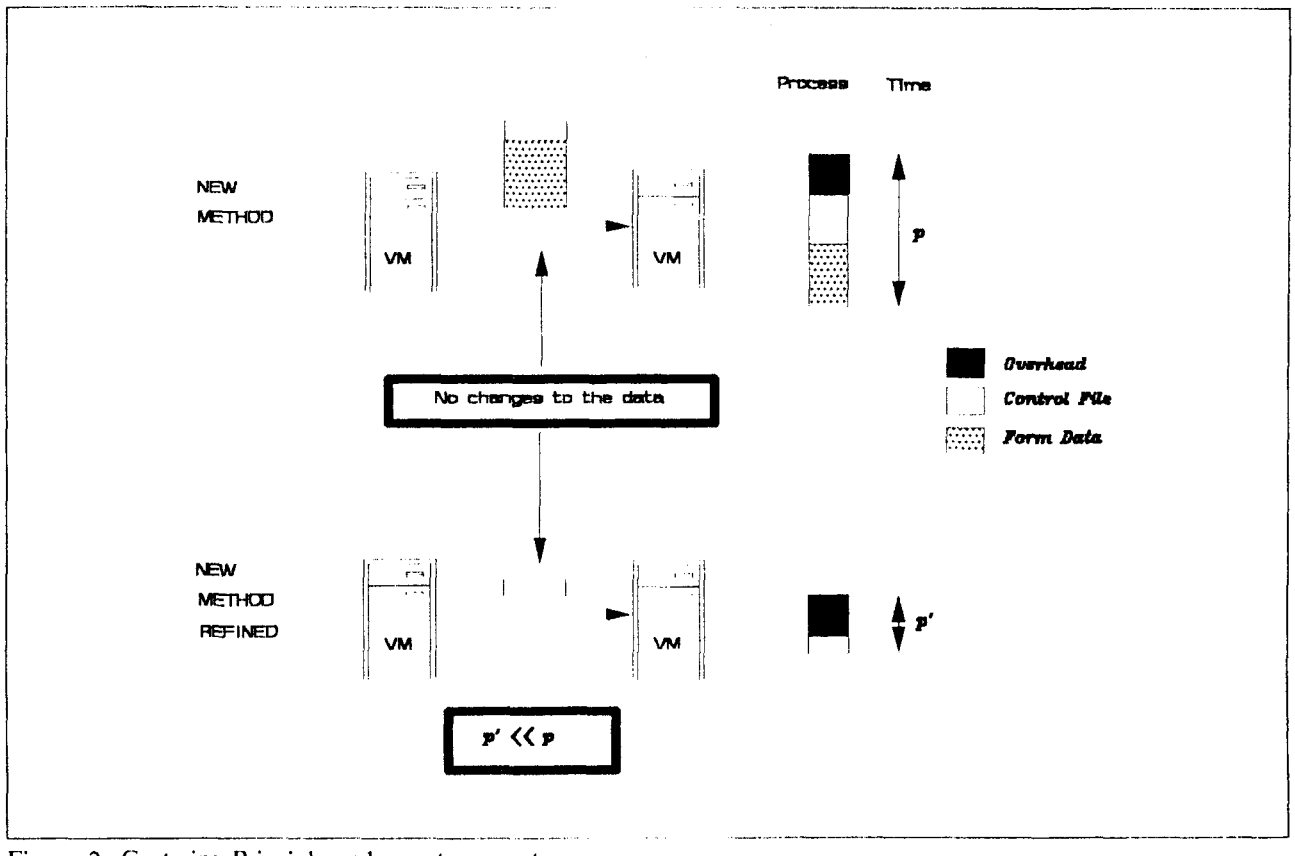

Figure 2. Centering Principle and remote requests.

- A further application of the centering principle within each transaction is the partitioning of the processes within a transaction into normal and exception partitions. Partitioning identifies the normal paths through the process and the unusual paths (exception partition) and focuses on the former from a performance point of view.

4. **The Parallel Processing principle states that processing should be partitioned into real or apparent concurrent processes provided that the benefit outweighs the communications and resource contention overheads.**

   - Some of the dominant transactions require the client to communicate via IUCV with a data server followed by an authorization server in a serial fashion. The authorization server is architected to a large extent on the basis that the data server has completed its processing successfully.

     To involve the data server in some sort of parallel processing with the client was not considered feasible because of the major architectural difficulties.

     However, in the case of the client-to-authorization server, a change has been prototyped where control is handed back to the client at a much earlier stage after some preliminary processing for appropriate transactions (Approval and Final Approval) as shown in Figure 3 on page x.

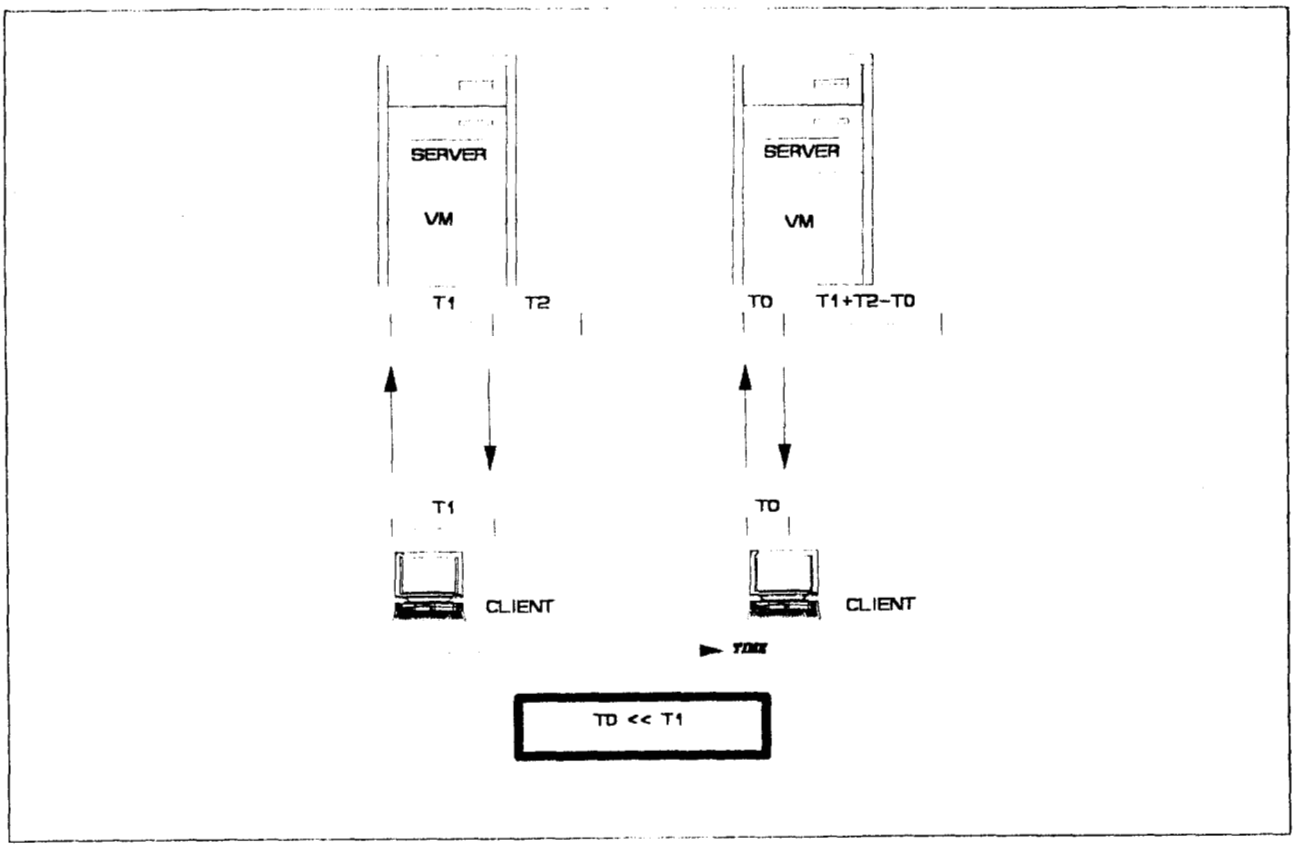

Figure 3. Parallel Processing Principle and Asynchronous request handling

The server continues processing asynchronously to the client. The ultimate outcome of the transaction is recorded, as usual, in a client log file.

From a client point of view, this reduces the transaction time very significantly (T1-T0) with little architectural impact on the server. Naturally, the improvement is maximised where there is no queue to the server and is diminished according to the number of requests in the queue.

5. **The Instrumenting principle encourages the instrumentation of the system as the means of measuring and controlling performance.**

   This is a control principle which does not directly improve software performance

   This principle was originally not part of the SPE methodology but was subsequently included because of its essential role in the performance effort

   Further discussion of this principle and the entire measurement process is continued later

Other important lessons were learned which were phased into the approach at different stages across the entire effort.

Measurements

For existing systems, measurements are the key to success. They provided us with an execution knowledge of the system enabling us to model the system. This model in turn allowed us to decide on the types of change required. Measurements were also the key to understanding the success or failure of the changes.

Experience and results have illustrated a number of important lessons:

1. Firstly, the importance of a proper measurement system is apparently not obvious to most people. Often developers express a sense of incredulity when asked if the system they had developed had been measured [5].
2. Where a satisfactory measurement process is not part of the approach, success is very dependent on intuition and luck and this is not a very scientific way to proceed.

3. The measurement process should use as many performance indicators as is practical to verify a performance prediction. A single indicator of performance (like response time) can be very misleading.
4. Early measurements of less than complete efforts are imperative. Even though these will often be contested on the grounds that further performance *tuning* will follow, they provide an early warning system which is often well-founded.

The measurement process itself is described later.

Language

Most of the product was written in REXX, the remainder in C. An important guiding principle which we adopted (and not just for language considerations) was that of benefit/cost maximization.

It's important to maximize the benefit/cost ratio where the benefit is the estimated improvement in performance to the customer, measured by the benchmark and the cost is the resource needed to develop and maintain that change.

In general, the CP and CMS commands and other external modules and not the REXX instructions were responsible for the substantial part of the product. Sample transactions show that the REXX instructions account for less than 20% of the total. In addition, poor performance was not caused by poor REXX coding but by lack of performance sensitivity in the original design stages with some notable exceptions. This was also borne out by informal discussions in the area of REXX performance with Mike Cowlishaw.

The main exceptions were the use of keyed arrays, which is a very slick way of searching for data, rather than the traditional binary search technique and the removal of Interpret statements to make the code compilable.

Conversion of the C code to REXX made good performance sense because of the way that the two languages interact differently with CP and CMS and some of this task has already been accomplished in Effort 1. This change from C to REXX has the additional benefit of easier maintainability.

Other improvements within REXX were better management of storage and in particular the dropping of storage when appropriate together with the complete specification of CP and CMS commands without ambiguity.

Changing REXX variable and procedure names, positioning of routines within the program, the use of one particular REXX built-in function over another were found to be examples of high cost - low benefit changes.

Exploratory Prototyping

We seized the opportunity to deviate from the standard practice of a documented low level design by prototyping the designed performance changes. This approach allowed us to measure progress at a stage much earlier than would have been possible with the traditional phased approach.

It also appears, based on a causal analysis of software defects found so far, that the prototyping made a very positive impact on the quality of the software shipped.

We strongly advocate prototyping as the best way to manage performance engineering of an existing system.

Other Items

For the first effort and because of resource constraints, measurements were confined to a 4381 processor running VM/SP5. This was restrictive and not very representative. The second effort has extended the measurement process to a 3083 running VM/XA and a 3090 running VM/ESA.

A comprehensive report [4] was created of the results of the first effort. This formed the basis of discussions which were held with the Performance team in the IBM VM Laboratory in Endicott, who reacted very positively to the depth of analysis and overall approach. The report has also been sent to all internal product sites to inform and encourage them to upgrade to the latest release.

The benchmark continues to be based on a single user and on a single processor for most of its transactions. The authors believe that some modeling of multiple users is a key area for the future which will help particularly in the area of capacity planning for our customers.

Functional development work of the product has continued alongside the performance effort. Both

teams have worked closely together with the performance team acting in an advisory role on the functional enhancements from a performance point of view. This close collaboration has been critical in the management of the performance work and has resulted in much greater sensitivity to performance within the entire group.

Measurement Process

The measurement process and its findings were both the guiding force behind the performance analysis along with being the ultimate arbiter on the success of the effort.

The adoption of the instrumenting principle at the very early stages of analysis, enabled us to isolate the major areas for each of the SPE design principles. The instrumenting principle is of greater significance for products that have been developed so far without the use of SPE and in a sense replaces the type of modeling, advocated within SPE for new products.

For this reason, the modeling effort is reduced and the measurement effort increased.

In this way, SPE is still very applicable to existing products but with shifts in emphasis when compared to new products.

1. **Performance Refinement**

   The initial stages of the measurement process involved a breakdown of the dominant transactions into their sub-components. These sub-components then became the subject of analysis through a limited set of unsophisticated measurements of virtual CPU time, Start I/O and response time. This provided us with important initial execution data of the system.

   In this way, the refinement provided us with a type of informal software execution graph of each of the transactions, a form of analysis advised under SPE.

   Of course, the REXX language with its rich tracing functions and its end-user friendliness, lends itself very well to this type of approach.

   This refinement pointed to those areas that should be concentrated on , which together with the other guiding principles (SPE and Benefit/Cost) already referred to, became the basis of the performance design changes.

2. **Prototyping**

   The main features of the proposed design changes were prototyped at a very high level and a new set of measurements obtained. These measurements formed the basis of the target objectives for each transaction within the benchmark which together with the design changes constituted the initial design document. This was subsequently approved by a selected list of external and internal reviewers.

   This allowed our customers an early indication of the magnitude of the performance improvements that could be anticipated and an incentive to agree to the resource investment.

   In this fashion, the refinement and limited prototyping provided us to a large extent with the necessary data to define the performance specification. The same type of data collection is also advocated under the SPE methodology, although the manner of collection is naturally different for new software systems.

   As part of the second design stage (referred to in the IBM phased approach as low level design) the prototyping exercise was continued at a lower level with the prototype being more closely aligned to the ultimate implementation.

   The prototyping exercise was really a prerequisite to the measurement process and they complemented each other very successfully.

   In a few instances predictions were made for some of the performance metrics based solely on the software execution graphs of the transactions. These predictions were then compared with actual results from the prototyping exercise and were used as a theoretical validation of the prototyping results.

   Basic measurements were periodically taken during the prototyping and modifications made where there were any deviations from the objectives. This design stage became a highly iterative process and emphasised the engineering approach to the whole problem.

3. **Data Collection**

   Even though the instrumentation was an integral part of the entire performance development cycle, it wasn't until the changes had been

designed and implemented that the more controlled measurement experiments were conducted using acquired customer data.

The entire measurement process is a complex one where there are so many contributing factors. We adopted a number of approaches to make it as realistic as possible within the working constraints. We concede that further enhancements are both desirable and necessary.

- Probes were inserted at appropriate parts of the end-user and service machines to track the metrics which included Virtual CPU time, Start I/O , Response time, free Virtual Storage and System Load. These probes were positioned to capture the metrics for

  a. The Total End-User Component which includes the waiting for a service machine to respond (a)

  b. The Total Service machine Component for both servers (b)

  c. The Interface part of the Total End-User Component (c, c < a, a-c < b)

- The system was triggered once certain initial conditions had been set up. These conditions were based on varied customer input. They included directory size, number of forms in progress, sizes of critical control files which were typical of a customer installation.

- Measurements were always conducted on both the new and old implementations in a number of ways:

  a. An old and new installation was set up on the same CPU and both were triggered simultaneously for a given set of benchmark measurements

  b. On other occasions, measurements of the old and new implementations were interwoven in the following manner - (old, new, old, new, old, new)

  c. All controlled measurements were run at off-peak times

4. **Interpretation and Evaluation**

Existing tools were used and new ones developed to enhance the measurement process.

- KEYPLAY is an IBM internal use tool which runs on OS/2 and executes a set of pre-defined keystrokes on a host machine. KEYPLAY has been used to execute the defined benchmark usually at a deferred point in time (off-peak), switch between different systems (old and new) dynamically, collect the results and invoke the other developed tools to analyse the results.

  KEYPLAY has been instrumental in providing a fully automated measurement process where it can be triggered during off-peak working hours and the following morning a summary of the results taken at off-peak is available on the disk of the requestor. The interpretation of and judgements about these results is still an important and necessary follow-up step.

- We have also developed extensive REXX tools to analyse the collected results.

- The results over a number of runs of the benchmark are treated as follows

  - The lowest and highest 10% of the runs are ignored leaving the middle 80% for interpretation in order to weed out extreme results.

  - This remainder is averaged and a comparison made between the old and new implementation.

  - Occasionally, we measure a control which is identical within both the old and the new implementations and normalize the results with respect to this control. Both the normalized and the unnormalized results are then interpreted and compared.

- Interpretation of measurement data is something which improves with performance analysis experience, familiarity with the actual task of data interpretation and knowledge of the software under investigation. The key is to treat results with caution and respect and the goal is to try to get reasonable consistency in your results.

- An important point to look out for is perturbation of the results by the measurement system itself. This is best checked by comparing the results of the probed system with the system without any probes.

- Management of the vast amounts of measurement data is important. We used a summary file to reference the data

belonging to a particular run of measurements which held key information about that run.

Results

The targets prototyped for the the original performance effort (Effort 1) were based on the three metrics of elapsed time, virtual CPU time and start I/O. These metrics were used for all the benchmark transactions throughout both performance efforts as a means of gauging our success.

We have represented a summary of the results in the following diagram (please refer to Figure 4 on page xvi and Figure 5 on page xvi) for both the end user and the service machines as follows:

1. Prototyped target results for Effort 1.
2. Actual achieved results for Effort 1.
3. Prototyped results for Effort 2 + actual achieved results for Effort 1.

A more detailed account of the actual results from Effort 1 is contained in a separate report [4].

The results are presented as a % reduction on the base at the start of Effort 1.

Our approach has led us to target a subset of the benchmark transactions. However the charts show the percentage reduction, in each metric, over the entire benchmark and not just over the targeted transactions. The reductions in the targeted transactions had to be higher to achieve this overall result. For example, the targeted transactions on the service machines had to achieve a 34% reduction in order to achieve the 31% overall reduction.

The main features of the results are that

1. End-User Response time over the entire benchmark has been reduced by 24 % and further prototyping indicates that we can achieve an overall reduction of nearly 50%.
2. We have achieved over 50% reduction for the service machine transactions and early prototyping has indicated that this reduction could be close to 70% when we have concluded the current effort (EFFORT2).

Conclusion

The main conclusions of the approach are :

1. A number of SPE methods can be applied, with significant success, to existing software products. This is particularly true of a REXX product which lends itself to in-depth analysis.
2. Prototyping is very necessary in predicting performance results. Prototyping also had a significant impact on the quality of the software shipped.
3. We can not over emphasize the importance of measuring results from an early stage in the development cycle. Constant re-measuring of results ensures that performance degradation is not allowed to creep into the project at any stage. REXX myths which had been presented to us as ways to improve performance, eg. Code tuning, were discarded by the measurement approach.
4. The key to finding what works on your product is through study of the SPE methodologies, analysis of the areas of your product where they can be applied and then measurement of the results that can be achieved to determine their cost effectiveness.

Apart from the significant performance improvement, the drive for improved product performance has also produced the following :

1. Performance culture established in the development group. It is important to recognise that alongside the performance work, functional development of the product has continued often in similar areas. It is testimony to the change in performance culture within the entire group, that this has been a relatively smooth collaboration.
2. Initial feedback from customers has shown a marked improvement in satisfaction with the performance of the product.
3. Performance guidelines established for the Laboratory.

4. Performance engineering, as part of the development cycle, highlighted within the group and the Laboratory.

In retrospect, the key to success has been in the overall approach. The use of SPE principles as a guiding force, the adoption of an exploratory prototyping approach together with a significant investment in the measurement process have been the critical success factors. The engineering concepts of design, measurement and assessment in an iterative fashion, have been the kernel of the entire approach.

In conclusion, we have proven that the use of SPE methodologies together with our own methods to improve the performance of an existing REXX product were both worthwhile and practical.

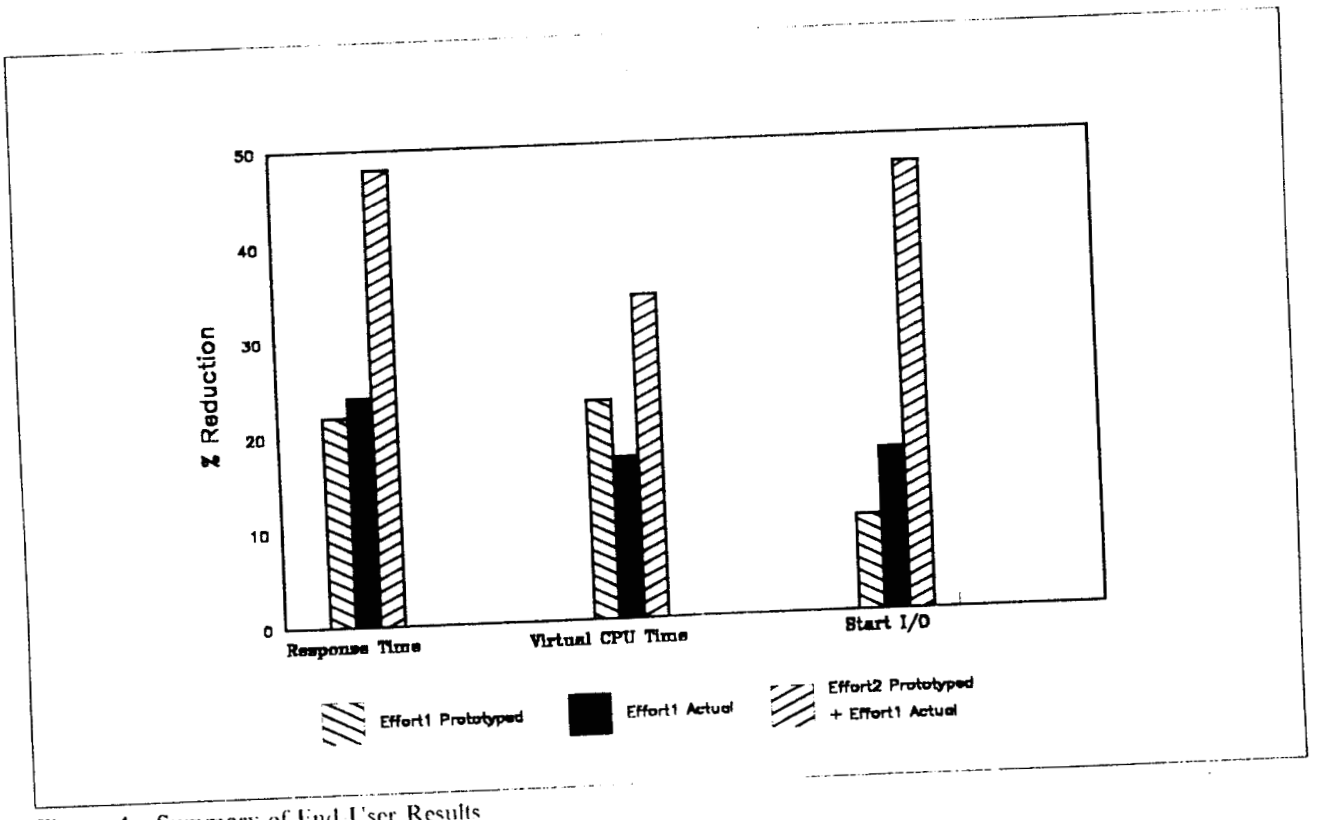

Figure 4. Summary of End User Results

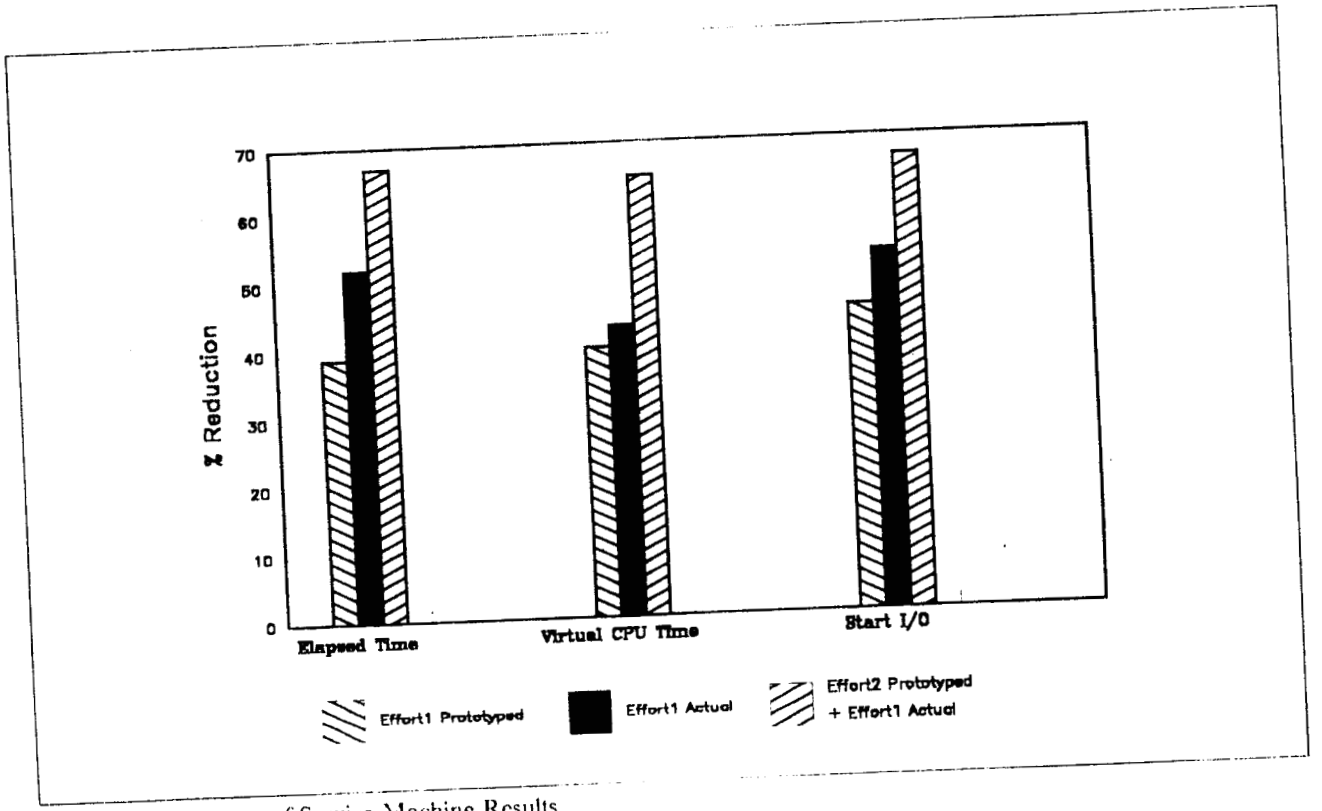

Figure 5. Summary of Service Machine Results

References

[1]Connie U. Smith, "Software Performance Engineering", Proc. Computer Measurement Group Conference XII, Dec 1981, 5-14

[2]Connie U.Smith, "Performance Engineering of Software Systems", Addison-Wesley, 1990

[3]Connie U. Smith, "Who uses SPE?", CMG Trans., Spring 1988, 69-75

[4]P. Meehan, IBM Internal Use document, ECFORMS Release 3.0 SPE 1 Performance Report, June 1991

[5]T.E. Bell and A.M. Falk, "Performance Engineering: Some Lessons from the Trenches", Proc. CMG 87, Orlando, Florida, Dec 1987

Other References

Gordon E. Anderson, "The Coordinated Use of Five Performance Evaluation Methodologies", CACM, 27,2 Feb 1984, 119-125

C.T.Alexander, "Performance Engineering: Various techniques and Tools," Proc. CMG Dec 1986, 264-267

P.J.Jalics, "Improving Performance the Easy Way", Datamation, 23,4, Apr 1977, 135-148

Connie U. Smith, 'General Principles for Performance Oriented-Design', Proceedings CMG 87, Orlando, Florida, December 1987, pp 138-144

Gwen A. Morrison, IBM Corporation, 'Performance for a large, complex application', Proc CMG 86, Dec., 1986, 316-320 Proceedings CMG 87, Orlando, Florida, December 1987, pp 138-144