The Association of System Performance Professionals

The **Computer Measurement Group**, commonly called **CMG**, is a not for profit, worldwide organization of data processing professionals committed to the measurement and management of computer systems. CMG members are primarily concerned with performance evaluation of existing systems to maximize performance (eg. response time, throughput, etc.) and with capacity management where planned enhancements to existing systems or the design of new systems are evaluated to find the necessary resources required to provide adequate performance at a reasonable cost.

This paper was originally published in the Proceedings of the Computer Measurement Group's 1982 International Conference.

For more information on CMG please visit http://www.cmg.org

SOFTWARE PERFORMANCE ENGINEERING

Connie U. Smith
Computer Science Department
Duke University
Durham, NC 27706

1. EXTENDED ABSTRACT

1.1 MOTIVATION

Software performance engineering (SPE) is a discipline used throughout the software life cycle, from early design stages through implementation and maintenance stages, to ensure the satisfactory performance of large scale software systems. It is a design-based approach which incorporates an analysis of proposed software designs in order to identify those designs which, when implemented, will result in acceptable performance.

It is important for increased leverage and productivity to begin the performance engineering early in the software life cycle. By identifying a suitable design and implementation strategy prior to coding, performance can be orders of magnitude better than that of software with an inappropriate design that is "tuned" to improve performance after implementation. Once the fundamental design decisions have been made it is generally infeasible to modify them later.

The second advantage of software performance engineering is productivity. A critical stage in the software development process is the testing stage. Poor software performance at this stage reduces the productivity of software developers since the amount of time required per test is unnecessarily high. Likewise, a substantial amount of software developers' time may be required to correct performance problems which require code modification and thus re-testing. Preventing these performance problems improves the software development process.

There are many other benefits to the use of SPE. It is typically quite cost-effective; that is, the cost of conducting the analyses at appropriate life cycle stages is more than offset by the performance savings from the early detection and correction of bottlenecks. If the design yields satisfactory performance initially, the additional effort required for performance engineering is minimal. It is much like "performance insurance."

Performance engineering has the greatest impact when applied to large scale software systems, but it is effective for systems of any size (which have specified performance goals). It is applicable to user-level software such as management information systems or decision support systems, to system support-level software such as data management systems, to system level software such as operating systems, and even to hardware levels for machine architecture and instruction implementation issues.

The techniques have been developed experimentally. Software systems have been studied throughout the life cycle and the performance engineering discipline has been formulated. It includes techniques for resolving typical problems. The accuracy of the performance analysis techniques applied in the early design stages is sufficient to identify bottlenecks prior to implementation [SMI79,SMI81]. The estimates of resource requirements in early design stages are sufficiently accurate to predict the limiting resource usage patterns [SMI82]. The accuracy of predictions improves as implementation proceeds and better estimates become available.

1.2 SYNOPSIS

The performance engineering process begins in the requirements analysis stage of the software life cycle. An initial study determines the feasibility and desirability of the proposed requirements, of the software functional architecture and, if appropriate, the hardware/software configuration required to support the new system. As the development proceeds and design and implementation strategies are resolved, additional analyses are conducted to study the current version of the software, to revise the predictions, and to assess performance impact. Later in the life cycle during the maintenance and modification stage, proposed enhancements are evaluated using techniques similar to those used during development. Data gathered throughout the life cycle is used for future capacity planning.

Data required for the initial analyses is gathered in performance walkthroughs. Models of the software are then constructed which include information on the workload, the software structure, and the execution environment. The techniques used are matched to the accuracy of the data. During early design stages when the data is imprecise, graph-based models are used to rapidly obtain approximations of performance. Later as the accuracy of the data improves, more sophisticated queueing network models are employed to yield better approximations of performance. Similarly the sophistication of the queueing network model representation increases as implementation proceeds.

The procedure for the detection and correction of bottlenecks in the design stage is

similar to that typically used in tuning. The bottleneck resource (CPU, I/O, memory, etc.) is identified and the execution graph models are examined to identify the software components which have the greatest demand for that resource. The familiar 80-20 rule used in program tuning (≤20% of the code accounts for ≥80% of the resource demands) typically applies to large-scale systems as well. That is, less than 20% of the programs in the system are generally the critical components. Those components are then examined and optimized.

As stated earlier, there is more flexibility in the optimization of systems at the design stage. Program enhancements (after code is implemented) are typically constrained to revisions which perform the same function in a better way. At the design stage, however, it is possible to easily change the functions that the software will perform. In fact, users are often quite willing to modify the software requirements when the (performance) cost to implement them is high. Many "requirements" are actually "desires" and the users who specified them need sufficient information on their cost and performance impact in order to properly identify the set to be implemented. This procedure can significantly improve both the resulting performance of the software and the productivity of the development team.

1.3 PERTINENCE

This type of evaluation is routinely done in engineering disciplines. Once a design is proposed, a thorough "quality" evaluation (which incorporates a wide variety of analyses) is performed to fine tune the design prior to construction. Performance of software systems is an important aspect of quality and thus this type of analysis should be included in the design stage. Other aspects of software quality should also be included such as reliability, testability, and durability (maintainable and modifiable). It is currently extremely expensive to develop and maintain software systems. Thus, performance engineering in particular and quality engineering in general are becoming increasingly important aspects of software development.

REFERENCES

[SMI79] Smith, C.U. and J.C. Browne, "Performance Specifications and Analysis of Software Designs," Proc. Conference on Simulation Measurement and Modeling of Computer Systems, Boulder (August 1979).

[SMI81] Smith, C.U., "Increasing Productivity by Software Performance Engineering," Proc. Computer Measurement Group XII, New Orleans (December 1981).

[SMI82] Smith, C.U. and Browne, J.C., "Performance Engineering of Software Systems: A Case Study," Proc. AFIPS National Computer Conference, Houston (June 1982).