*Proceedings of the 7th International Conference on Applied Informatics*
*Eger, Hungary, January 28–31, 2007. Vol. 1. pp. 295–301.*

# Performance evaluation of large-scale data processing systems

**Attila Adamkó, Mátyás Arató, Gábor Fazekas, István Juhász**

Department of Information Technology, University of Debrecen
e-mail: {adamkoa,arato,fazekasg,pici}@inf.unideb.hu

**Abstract**

Recently, integrated data processing systems have obtained more and more highlight as a result of their continuously increasing strategic role. The fast penetration of these systems requires new approaches in examining and measuring the performance with main respect to the utilization and use of various database systems (and technologies) with statistical methods. In the analysis we would like to find the characteristics which could be used to support the performance evaluation from various aspects. Hereinafter, we present the most useful approaches used for this evaluation process:

- Availability
- Reliability
- Advantages and disadvantages of release changes
- User satisfaction
- Validation
- System parameterization.

Based on these viewpoints we can find answers concerning efficiency, initiation time and performance information both in the present and in the future. Moreover we can compare these systems, and hopefully we will be able qualify these systems.

*Categories and Subject Descriptors:* D.2.8 [Software Engineering]: Metrics; C.4 [Performance of Systems]; G.3 [Probability and Statistics]

*Keywords:* Performance evaluation, measuring, metrics, data-processing systems

## 1. Introduction

As a result of the continuous evolution of data management technologies data processing systems comprise even more and more functionality. Therewith, the amount of stored data is constantly increasing and we would like to use tools that can manage and process this huge amount of data.

Nowadays several different integrated data processing system are available and their strategic role is continuously increasing. These systems are mainly using the Enterprise Resource Planning (ERP) term. A key ingredient of most ERP systems is the use of a unified database to store data for the various system modules.

The fast penetration of these systems highlights the need of deeper investigation on this area. In this paper we would like to find the determinant characteristics of performance evaluation of these systems and to find answers concerning efficiency, performance and reliability.

## 2. Performance

To start the investigation we need to answer the first and topmost question:

What we mean under the term performance or a little refined, how could we interpret the performance?

The main problem is that there is no exact definition. The researchers are using their own notations and definitions to describe their domain and interpretations. Consequently, we need to establish our own viewpoints. The second problem is that there is no uniform approach which could be used for measuring the performance.

Moreover, we could interpret the notion of performance at two different sides. On the one hand, we could make considerations on the data processing side analysing the quality of data handling processes. On the other hand, we could investigate the business logic side analysing the correspondence to the business needs and requirements. In both cases it is important to pay particular attention to the relevant factors, like reliability, availability and rapid answer.

### 2.1. Measurement

After this short introduction, we need to establish the basis of our investigation by answering the following questions:

- What to measure?
- How to measure?

The answer of our first question requires the establishment of viewpoints that consider both quality factors and software properties. However, these two characteristics take into the picture the notion of metrics. In order to use metrics we need information about the topically significant parts. What is significant depends on the data available and the measuring objectives. Naturally, most of us connect the metric term with software quality at fist glance, but in our case we need to measure the performance. Nevertheless, a good metric owns the following attributes. It is:

- exactly defined – known what is measured,

- objective – everybody can take the measurement,
- validated – measures what it should,
- robust – insensitive of changes of not relevant factors,
- easy to compute.

A software quality factor is a non-functional requirement for a software program which is not called up by the customer's contract, but nevertheless is a desirable requirement which enhances the quality of the software program. When evaluating a system's performance we need to find similar quality factors but relevantly to the performance. Moreover, these factors cannot be measured because of their vague description. Some of them cannot be evaluated in its own right. However there are related attributes which can be measured indeed. For us, it is necessary to find measures, or metrics, which can be used to quantify them as non-functional requirements. In our point of view the following relevant factors could be identified:

- availability – the time that the system is entirely available for its intended purpose. The most important factor will be the number and the length of downtime (either planned or unplanned).
- user satisfaction – a subjective number, describing user feedback based on the functionality of the system.
- validation – interpreted in multidimensional:
  - data side: beyond consistency, the data could correspond for the business logic,
  - system side: the system could be adapted through its parameterization to full fit its requirements.
- system parameterization – how could be influenced the systems quality through short-term and long-term parameterization.
- reliability.

The last factor, reliability could be also interpreted in a multidimensional manner. At one side, it could represent confidentiality, meaning that my data is accessible only for me. At another side, it could mean integrity, or inter-operability between layers of the software.

## 2.2. Reliability metrics

Reliability is a complex concept that should be considered at the system level and should be specified as a non-functional requirement. Reliability metrics are used to measure the frequency of failures and to predict the likelihood of a software failure.

The following metrics have been used for specifying software reliability and availability:

- POFOD – Probability of failure on demand: The likelihood that the system will fail when a service request is made.
- ROCOF – Rate of occurrence of failure: The frequency of occurrence with which unexpected behavior is likely to occur.
- MTTF – Mean time to failure: The average time between observed system failures.
- AVAIL – Availability: The probability that the system is available for use at a given time.

To predict the system's reliability there is no available abstract method. Therefore, our investigation deals with the analysis of the measured data. Using a data-oriented approach, we need data sources like log files, output files, etc. The necessary data collected by an accounting process. This data could be interpreted a definite way influenced by the requirements of the management. Generally speaking, the company's profile together with the management's requirements determines the desirable data or on the contrary we are focusing on the information that the management would like to see. This could be done because there are no uniform approaches available so we could take into account the business aspects influencing the measurement.

## 3. Statistical testing

To validate that the system meets our requirement, we have to measure its reliability at first. In recent literature, we could find a proper method used in the validation process, called statistical testing.

The process of measuring the reliability of a system is illustrated in Figure 1.

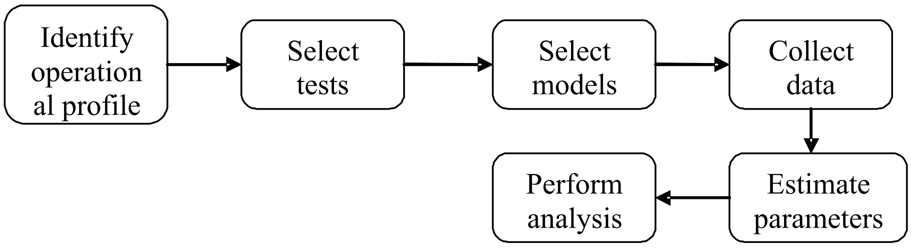

Figure 1: Statistical testing

This process involves six steps:

- identify operational profile – this profile reflects how it will be used in practice. It consists of specification of classes of input and the probability of their occurrence.

- select tests – we need to know the testing approach because it may influence how the failure data are used.
- select models – a model's applicability is determined by how well it accommodates a project's special feature.
- collect data – we need to construct test data that reflects to the operational profile using test data generators. We need a statistically significant amount of failures, of course
- estimate parameters – in this step we could choose from the three common methods like least squares, maximum likelihood and the method of moments.
- perform analysis.

The software execution environment includes the hardware platform, the operating system software, the system generation parameters, the workload, and the operational profile. Software reliability testing is based on selecting input states from an input space. An input state is a set of input variable values for a particular run. The input states chosen for test cases should form a random sample from the input state in accordance with the distribution of input states that the operational profile specifies. Once the operational profile is established, a procedure for selecting a random sample of input states is required, so that test cases can be generated.

A software reliability growth model allows project management to track the progress of the software's reliability through statistical inference and to make projections of future milestones. In our point of view, reliability means something like usability.

## 3.1. Model selection

There are several software reliability models available, like Musa Basic, Littlewood/Verall, etc. A model's applicability is determined by how well it accommodates a project's special feature. If the model is not fitting well, then we should switch to an alternative model and/or parameter estimation technique. Some software reliability modeling tools allow models to be combined, or to develop your own model.

A common approach for measuring software reliability is the use of an analytic model whose parameters are generally estimated from available data on software failure. For example let we choose a simple model, like Musa Basic. To estimate the reliability we need some parameters:

- Average total number of failures: $\mu(t)$
  Average refers to $n$ independent instantiations of an identical software.
- Failure intensity: $\lambda(t)$
  Number of failures per time unit, derivative of $\mu(t)$.

- $t$ may denote elapsed execution calendar or machine clock time

Musa's Basic model assumption is that decrement in failure intensity function is constant. Its consequence is that the failure intensity is function of average number of failures experienced at any given point in time (= failure probability).

$$\lambda(\mu) = \lambda_0 \left[1 - \frac{\mu}{\nu_0}\right]$$

- $\lambda(\mu)$: failure intensity.
- $\lambda_0$: initial failure intensity at start of execution.
- $\mu$: average total number of failures at a given point in time.
- $\nu_0$: total number of failures over infinite time.

Figure 2: Basic Musa model

We can see that there exist several extension points in this model. Moreover, this factor is only one of the above discussed factors. A deeper knowledge is needed to select the proper model and to estimate the correct parameters. We can use several data sources to help this parameter estimation but statistical methods are needed to assist in the selection of the correct values.

After all the measurements are made, we can characterize the behaviour of the system from this aspect and we can make some predictions. However, there are other relevant aspects which have to be measured to qualify a given system. Currently we have some imagination concerning these directions but further investigations are necessary.

## 4. Future work

In this preliminary experiment we observed that reliability is an important factor in the performance of these models. The primary challenge is to identify the relevant factors and an extensive analysis of operational profiles.

Further experiments are needed involving new cases and problems to support this conjecture. Also it is an exciting challenge to implement improvements in the software prediction process based on these ideas. These time-based processes are related with each other and our goal is to make considerations about it using differential equations and time-series. When we find proper models we could make statistical computations to predict for example the alteration of response time if overall system load goes from the normal 80% over 90%. In other cases we could make predictions about system performance changes and similar effects.

## References

[1] Musa, J. D., Iannino, A., Okumoto, K., Software Reliability: Measurement, Prediction, Application, *McGraw-Hill,* (1987).

[2] Littlewood, B., Theories of Software Reliability: How Good Are They and How can They be Improved?, *IEEE Transaction Software Engineering,* SE-6(5), (1980), 489–500.

[3] Kuo, S. Y., Hung, C. Y., Lyu, M. R., Framework for Modeling Software Reliability, Using Various Testing-Efforts and Fault Detection Rates, *IEEE Transaction on Reliability,* 50(3), (2001), 310–320.

[4] Wood, A., Predicting Software reliability, *Computer,* 29 (11), (1996), 69–77.

[5] Goel, A. L., Okumoto, K., Time-dependent error-detection rate model for software and other performance measure, *IEEE Transaction for Software Engineering,* 11(12), (1985), 285–306.

**A. Adamkó, M. Arató, G. Fazekas, I. Juhász**
Department of Information Technology
University of Debrecen
H-4010, P.O. Box 12, Debrecen
Hungary