Measuring Component Performance Using A Systematic Approach and Environment

Jerry Gao, Ph.D., Chandra S. Ravi, and Espinoza Raquel
San Jose State University, email: gaojerry@email.sjsu.edu

Abstract: As more third-party software components are available in the commercial market, more people begin to use the component-based software engineering approach to developing component-based programs. Since most e-business and e-commerce application systems are net-centered distributed systems, customers usually have very restricted system performance requirements. Hence, performance testing and evaluation of software components becomes a critical task for component-based software. Although a lot of research efforts have been devoted to analysis and design methods of component-based software, only a few research papers address the issues and systematic solutions to performance testing and measurement for modern components. This paper proposes a systematic solution and environment to support performance measurement for software components. The objective is to provide a plug-in-and-measure approach to evaluate component performance, including functional speed, throughput, availability, reliability and resource utilization. It reports the development effort on constructing a distributed performance evaluation environment for software components based on a set of well-defined performance evaluation metrics and techniques. In addition, some application examples and case study results are reported.

Keywords: Component-based software engineering, performance measurement, performance metrics, performance evaluation, and software component evaluation environment.

1 Introduction

Widespread development and reuse of software components has been regarded as one of the next biggest phenomena for software industry. As more third-party software components become available in the commercial market, more people begin to use the component-based software engineering approach to developing component-based programs. Today most e-business and e-commerce application systems are net-centered distributed systems, customers usually have very restricted system performance requirements. System performance testing and evaluation therefore becomes critical in quality validation. Since component-based software performance is dependent on the performance of involved components, component performance evaluation becomes a critical task for component-based software projects. Although recently there are many technical papers addressing issues and solutions in analysis and design methods of component-based software, only a few published papers focus on performance testing and measurement issues for components and component-based software [3][4][7][10].

In the past decades, a lot of research efforts have been devoted to program performance evaluation and analysis in a black-box approach. Many published articles proposed performance evaluation metrics focusing on different aspects of system performance, such as speed, latency, throughput, load boundary, reliability, availability, and scalability [1][2][4][7][8][9][10][11][13][14]. There are a number of papers discussing performance evaluation models for conventional software systems [2][7][10][11][15]. In addition, there are a few papers addressing the systematic performance evaluation for software systems [13][14]. However, most existing work only address performance evaluation and measurement at the system level using a black-box approach, in which the system is considered the black-box, and performance evaluation is measured in a black box. view. Hence, component performance evaluation is not emphasized and component performance data is not collected. As pointed by Jerry Gao et al in [6], this traditional performance evaluation approach is not good enough for component-based software engineering because it is very difficult for engineers to isolate and discover components that causes system performance issues. He points out two reasons:

- Component performance validation for third party components must be performed prior to their use. Since most reused components are not developed to facilitate component performance validation and measurement, it is impossible to insert component performance probes inside commercial components. It is, therefore, costly to collect and conduct performance testing and measurement for components and component-based programs.
- Since component-based software systems are constructed based on third party and/or in-house build components, component-based software performance is dependent on the performance of involved components. This suggests that system level performance evaluation must be performed in a gray-box approach based on involved components so that any component performance issues in a targeted context and environment could be easily detected and isolated whenever a system performance issue is discovered.

In the real world, engineers encountered a number of issues relating performance evaluation for components and component-based software:

- It is not easy to understand component performance due to the lack of component performance reports and evaluation data for reusable components, including COTS components.
- It is very hard to detect and isolate performance problems for components in component-based software due to the lack of component-based software performance evaluation models and systematic solutions.
- It is very difficult to evaluate the performance of third party (or in-house) components in a targeted reuse context environment because they are not constructed to facilitate component performance evaluation.
- It is very costly to measure component performance in a reuse context due to the lack of performance evaluation tools that provides engineers a plug-in-and-measure environment.

As pointed out by Jerry Gao et al. [5][6], today engineers also have encountered a number of challenges in performance testing and measurement of component-based software:

- How to define and select component-based performance evaluation models and measurement metrics?
- How to establish a product-oriented or enterprise-oriented performance testing and evaluation environment for component-based software in a rational way?
- How to collect and monitor component performance data with minimum cost and overhead?

This paper proposes a systematic solution and environment to support the performance measurement for software components. The objective is to provide a plug-in-and-measure approach to evaluate component performance, including functional speed, throughput, availability, reliability and resource utilization. It reports the development effort of a distributed performance evaluation environment for software components based on a set of well-defined performance evaluation metrics and techniques. In addition, some application examples and case study results are reported. The structure of the paper is organized as follows. Section 2 reviews software performance metrics and discusses three special performance metrics for components. Section 3 proposes a systematic solution and environment to measure software component performance based on a set of well-defined metrics. Section 4 discusses the performance tracking and collection techniques. Section 5 reports our application example with a simple case study. Finally, the conclusion remarks are given in Section 6.

2 Understanding of Software Performance Testing and Evaluation

System performance testing and evaluation refers to testing activities and evaluation efforts on validating system performance and measuring system capacity. Today, performance testing and evaluation is a very critical step in a software testing and quality control process. It has **three** major objectives. The **first** is to confirm that the specified system performance requirements for a given product are satisfied. The **second** is to find out the product capacity in system performance to answer the expected questions from marking people and customers. The **third** is to discover the performance issues, such as performance degradation, performance improvements, and performance bottlenecks, in the given product release to help the development teams to identify of their causes.

Good performance testing and evaluation results provide solid system and component performance data to marketing people and customers concerning the product capacity of each system configuration. Engineers not only need performance data and reporting for the system but also its components, including functional processing time, throughput, availability, reliability, and scalability. Typical performance metrics are given as follows:

- ***Processing time:*** Validate the processing time at the component-level and system-level including the maximum, minimum, and average response time.
- ***Throughput:*** Checking system throughput enables us to understand various system throughputs in network communication, application processing, functional tasking, and business transaction. For a component-based application system, attention is paid to the minimum, average, and maximum system throughputs at the component-level and system level for each type of media inputs. The focus here is to determine

how much input have been processed successfully, and how much inputs have not been processed or abandoned.

- ***Availability:*** Checking component availability allows us to see the degree of component availability in the aspects of network, computer hardware & software, and application functions and services. For a component-based application system, it is important to evaluate the availability of the given system and its components during the given performance testing time period.
- ***Reliability:*** Checking component reliability allows us to determine how reliable a given component is in a given system environment with the specified system loads during a test period. For a component-based application system, it is important to evaluate the reliability of system components and system in a whole during the given performance testing time period.
- ***Scalability:*** Checking component scalability helps test engineers to determine how well a given component can be scaled-up on network traffic, server load and throughput, data and transaction volumes, application processing capability, and the supporting number of clients and concurrent users. To do this, we need to check the speed-up and threshold on component performance, and the component capacity threshold and improvement on component load and throughput.

The detailed common performance metrics applicable to modern components are given in [6]. The rest of this section discusses three special component metrics. They are component throughput, reliability, and availability metrics.

2.1 Throughput Metrics

Throughput metrics usually is developed to measure the successful rate of a software component to process incoming events, messages, and transaction requests in a given timeline. For example, we can define a transaction-oriented throughput metric for software components as follows:

> ***Definition:*** The ***transaction throughput*** of a component C_k for transaction type ***TR*** in a given performance test period T_p is equal to ***m***, where ***m*** is total number of successfully processed transaction requests. The formal metric is given below.
>
> ***Throughput(Ck, Tp, TR) = m***

During component performance testing and evaluation, engineers may apply a number of test sets to find out the maximum, minimum, and average transaction throughput for each type of transactions. Using the transaction throughput of components, we can define transaction throughput rate below.

> ***Definition:*** The transaction throughput rate of a component C_k for transaction type ***TR*** during a given performance test period T_p is the ratio of ***m,*** the total number of successfully processed transaction requests to ***n,*** the total number of received transaction requests. The formal metric is given below.
>
> ***Throughput-Rate(Ck, Tp, TR) = m / n***

When the throughput rate reaches 1, component C_k has the highest successful rate in processing transaction requests in ***TR*** category. Based on the same idea described here, other throughput metrics can be defined to evaluate the processing rate of incoming events and messages for software components.

2.2 Availability Metrics

Availability metrics is a standard metric for system performance measurement. According to [10], it is defined as the ratio of system uptime to the total system evaluation time, including system uptime and downtime. Based on this, system unavailability can be defined as 1 – availability. Therefore, system availability can be formally defined as follows:

system_availability = system_uptime / system_evaluation_time

Clearly, this metric is very simple and easily understandable. It can be applied to measure component availability because each component can be considered as a black box. However, evaluating component availability with this metric has several problems. They are summarized as follows:

- This metric cannot provide the detailed indications between the component availability and component supporting functions. Therefore, it is not useful for performance problem analysis, tuning and debugging.
- This availability metric is not applicable to software components with the fault-tolerant features. Today, many software application systems, such as web-based information systems, e-commerce systems, and defense systems, require high-availability components. How to measure their availability is a challenging issue for engineers.
- It is not easy for engineers to measure and monitor software component availability for a distributed component-based system unless some systematic solution is in place.

Here we propose two the component availability metrics to address the first two problems described above.

A) Function-Based Component Availability

Function-based component availability measures a component's availability based on its availability of supporting its system functions.

> **Definition:** The function-based availability of a component C_k in a system ***S*** refers to the ratio of its total available time $\mathbf{T_A}$ of supporting its system function feature F_j to the performance test period T_p, including both available and unavailable time to support function F_j. The formal metric is given below.

$$FComponent\ Availability\ (C_k, S, T_p) = T_A\ (C_k, F_j)\ /\ T_p$$

Like the standard availability metric, this definition is simple and straightforward. Unlike the standard availability metric, this metric focuses on the component availability of supporting a specific system function. This metric has two advantages. First, it is easy to measure, and its measurement results are useful for performance problem detection, analysis, and tuning. Second, it is easy to isolate and identify the availability problems at the component-level availability during performance testing and measurement at the system level.

B) High Availability (HA) of Components

To measure components with high availability requirements, we must define a metric to evaluate high availability of components. To achieve this goal, we first need to understand what is a highly available component.

> **Definition:** A high available component is an N-Cluster, in which N redundant components are actively running at the same time to support and deliver the same set of functional features as a black box.

Let's assume that $C_{HA} = \{C_0, C_1, \ldots, C_N\}$ is a cluster of N redundant components. Assume that there is a consistent mechanism to collect the perform results on component availability for each internal component. The high availability of an N-cluster component C_{HA} in a system S during the performance evaluation time T_p is computed as follows.

$$\textbf{Component-High-Availability}\ (C_{HA,}, S, T_p)$$
$$= \textbf{Havailable-time}(C_{HA}) \,/\, [\, \textbf{Havailable-time}\,(C_{HA}) + \textbf{Hunavailable-time}\,(C_{HA})\,]$$

Havailable-time (C_{HA}) can be computed based on a set of available time slots $\{ T_{1\,(HA)}, T_{2\,(HA)}, \ldots T_{m\,(HA)} \}$ for C_{HA} during the performance evaluation time T_p, where $T_{j\,(HA)}$ refers to a time slot in which at least one component in C_{HA} is active and available to deliver all functions. Similarly we can define ***Hunavailable-time*** (C_{HA}) where $\{T'_{1\,(HA)}, T'_{2\,(HA)}, \ldots T'_{n\,(HA)}\}$ refer to the time slots in which all components in C_{HA} are not available to support and deliver their functions.

$$\textbf{Havailable-time}(C_{HA}) = \Sigma_{j=1\ to\ m}\ T_{j\,(HA)}$$
$$\textbf{Hunavailable-time}(C_{HA}) = \Sigma_{j=1\ to\ n}\ T'_{j\,(HA)}$$

Figure 2.1 shows an example of a 3-cluster HA component and its available and unavailable time.

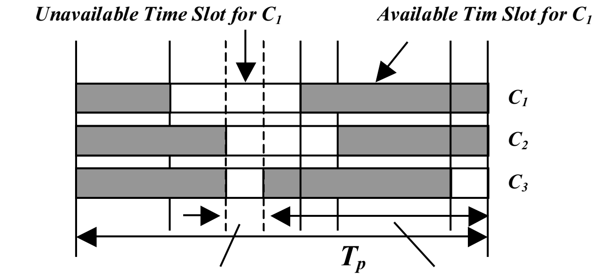

Unavailable Time Slot for C_1
Available Tim Slot for C_1
C_1
C_2
C_3
T_p
Unavailable Time Slot for C_{HA}
Available Time Slot for C_{HA}

Fig. 2.1: 3-cluster HA Component Available and Unavailable Time

2.3 Component Reliability

Although many different approaches and metrics have been proposed to measure system reliability, the common way is to evaluate the system reliability based on its reliability of service, in which a function *R(t)* is used to present the probability that service survives until time *t*. The reliability of the service is often characterized by specifying mean time to failure (MTTF) or mean time between failures (MTBF). In the computation of the MTTF or the MTBF, it is usually assumed that the exponential distribution best describes the possible occurrence of failures in the service. For different application systems, the reliability of service should be checked to make sure that the above definition is good enough to adequately support the evaluation of system reliability in a given application domain.

When we consider software components as a black box, many system-level reliability evaluation methods are applicable. Here, we define component reliability based on component uptime and downtime for services during a performance testing and evaluation time.

> **Definition:** The component reliability of a component C_k in a system during a time period T refers to the ratio of the total uptime of the component to the total time, including both available and unavailable time.

Based on this definition, component reliability can be evaluated easily if there is a systematic way in place to track component uptime and downtime. Let's assume that C_k is a non-HA component, and its component availability during time period T can be computed as follows, where T_p stands for the total performance evaluation time, and **up-time(C_k)** represents the up time of C_k during T_p.

$$\textit{Component-Reliability}_T(C_k) = \textit{up-time}(C_k) / T_p$$

For HA components, such as N+1-Clusters, we need a different metric to evaluate their reliability. The major question here is to show to evaluate the uptime and downtime of a HA component C_{HA} with N+1 redundant components. Here, two different methods are given. Figure 2.2 shows the difference between these two approaches.

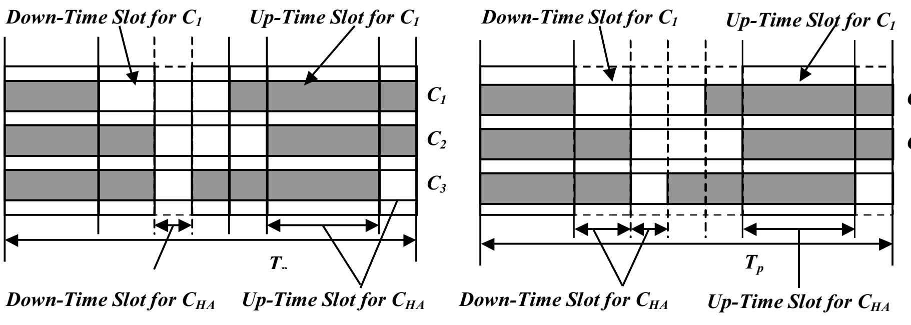

Down-Time Slot for C_1
Up-Time Slot for C_1
C_1
C_2
C_3
T_p
Down-Time Slot for C_{HA}
Up-Time Slot for C_{HA}
(A) Based on Single Failure Criterion
Down-Time Slot for C_1
Up-Time Slot for C_1
C_1
C_2
T_p
Down-Time Slot for C_{HA}
Up-Time Slot for C_{HA}
(B) Based on Service Availability

Fig. 2.2: Comparison of Two Approaches for Component Reliability

The first approach is based on the single failure criterion, that is, any component of a HA component having a failure of service will be considered as a HA component's failure. Therefore, the reliability of a HA component C_{ha} can be easily computed as follows, where ***up-time(C_{ha})*** represents the up time of $\boldsymbol{C_{ha}}$ during time period $\boldsymbol{T}$, and ***down-time(C_{ha})*** includes the down time and recovery time of $\boldsymbol{C_{ha}}$.

$$\textit{Component-Reliability}_T(C_{ha}) = \textit{up-time}(C_{ha}) / (\textit{up-time}(C_{ha}) + \textit{down-time}(C_{ha})) \\ = \textit{up-time}(C_{ha}) / |T|$$

To compute ***down-time(C_{ha})*** let's assume that $C_{ha} = \{ C_{ha1}, ..., C_{haN} \}$ is N+1 HA component. $\boldsymbol{T_{hai} = \{T_{(hai,j)} \mid j = 1,\ldots,m \}}$ is the set of downtime slots for $\boldsymbol{C_{hai}}$ in $\boldsymbol{T}$.

$$\textit{down-time}(C_{ha}) = \Sigma_i\, \textit{down-time}(C_{hai}) = \Sigma_i \Sigma_j T_{(hai,j)} \qquad (j=1,.., m, i=1,\ldots, N)$$

Notice that the Σ operation here represents the summation of all time slots by ignoring their overlapping time. The ***up-time(C_{ha})*** can be expressed in terms of ***down-time(C_{ha})***.

$$\textit{up-time}(C_{ha}) = |T| - \textit{down-time}(C_{ha})$$

The second approach evaluates the reliability of a HA component based on its service availability. Here, a failure of service for a HA component in a given time slot suggests that all components in a HA component C_{ha} have a failure of service in the same time slot. This implies that this approach uses a different way to compute *up-time(C_{ha})* for a HA component with N+1 redundant components. Let's assume that $C_{ha} = \{ C_{ha1}, ..., C_{haN} \}$ is N+1 HA component. $\boldsymbol{T'_{hai} = \{T'_{(hai,j)} \mid j = 1,\ldots,m \}}$ is the set of uptime slots for C_{hai} in T. Then, ***up-time(C_{ha})*** for $\boldsymbol{C_{ha}}$ is given below.

$$\textit{up-time}(C_{ha}) = \Sigma_i\, \textit{up-time}(C_{hai}) = \Sigma_i \Sigma_j T'_{(hai,j)} \qquad (j=1,\ldots, m, i=1, \ldots, N)$$

Notice that the Σ operation here represents the summation of all time slots by ignoring their overlapping time.

3 A Systematic Performance Evaluation Solution for Software Components

This section reports a systematic solution to support component performance measurement in a distributed environment. This solution is useful for component vendors to test and measure component performance, as well as for component users to validate component performance in component-based software using a client-server setting. The major objectives of this solution are as follows:

- Provide a common framework to support component performance testing and measurement. With this framework, engineers can convert software components to facilitate component performance evaluation with a consistent interface.
- Provide a consistent evaluation environment and mechanisms to measure component performance in a plug-in-and-measure approach.
- Provide a scalable, distributed performance evaluation environment that supports performance evaluation of components on any computer node over a network.

The solution consists of the three major parts:

- **Component Performance Evaluation Library:** The well-defined component performance evaluation library allows for constructing (or converting) conventional components into measurable components with consistent interfaces for performance evaluation.
- **Systematic Performance Evaluation Environment:** This environment is developed based on a set of well-defined component performance metrics, and performance tracking techniques.
- **Performance Tracking Methods and Formats:** Well-defined performance tracking methods and formats are useful to develop consistent performance tracking and evaluation mechanisms for software components.

3.1 Component Performance Evaluation Library

The component performance library is developed to provide engineers with a standard performance evaluation framework to support performance testing and measurement of software components and component-based programs. This library enables component developers to construct measurable components that facilitate performance measurement and to convert third-party components into measurable component by adding a consistent performance wrapper. The library provides two general functions for each software component:

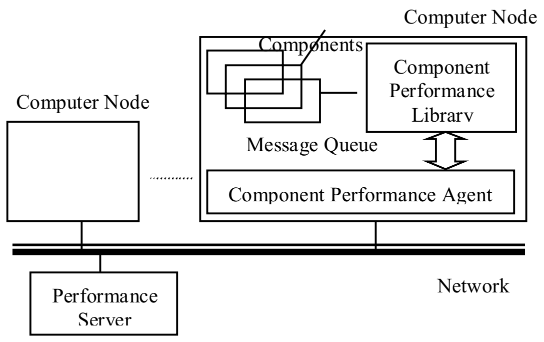
- Generate and collect different types of component performance traces using well-defined trace formats, and place them into a performance message queue.
- Compute component performance measures based on a set of pre-defined performance metrics.

Currently, the component performance evaluation library includes a super-class (known as performance class) and six derived classes. They are used for the measurement of component speed, resource utilization, transaction throughput, reliability, availability, and scalability. Another class supports the processing of performance message queues. They have been implemented as a component performance library based on the component performance metrics given in [6]. The details of this library as reported in [11].

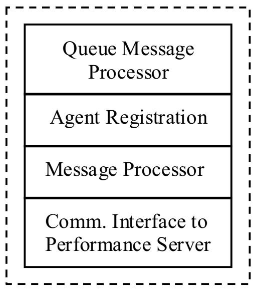
3.2 Distributed Component Performance Evaluation

Figure 3.1(a) displays the system infrastructure of a distributed component performance measurement environment. It supports performance testing and evaluation of software components and consists of a performance test and evaluation server, a number of performance agents, and a performance library within a client-server environment.

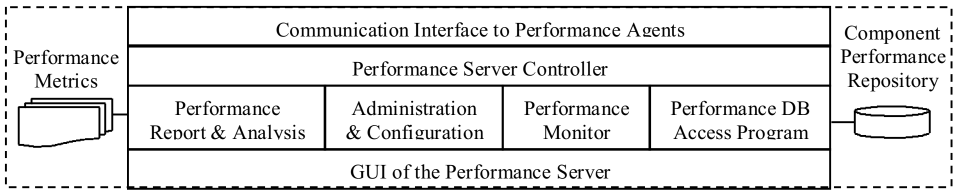
- ***Performance Test and Evaluation Server:*** The performance test and evaluation server manages and controls a central performance repository that stores various types of component performance trace records and evaluation results. The performance test server plays a control and management role in supporting performance trace collection, performance analysis and computing, as well as performance monitoring and reporting. The other function of the performance test server is to allow testers to control and configure performance tracking and testing features by communicating with performance agents on different computing stations over the network. Figure 3.1(c) shows the components of a performance test server. Through the GUI interface, engineers can communicate with performance agents on different computing stations to monitor the current component performance, query the agent status, and configure performance-tracking properties. The component performance analysis module analyzes and reports component performance in a graphic format.



(a) Performance Client-Server Structure



(b) Performance Agent Structure


Communication Interface to Performance Agents
Performance Metrics
Performance Server Controller
Component Performance Repository
Performance Report & Analysis
Administration & Configuration
Performance Monitor
Performance DB Access Program
GUI of the Performance Server

(c) Performance Server Structure

Fig. 3.1: Distributed Component Performance Evaluation Environment

- ***Performance Agent:*** On each network node (say a computer station), there is a performance agent that interacts with measurable software components through a dedicated performance message queue. It collects component performance data, and transfers them to the performance test server over the network using dynamic performance messages. The other function of a performance agent is to allow engineers to configure and set-up the required performance tracking and monitoring features by communicating with the performance test server. Figure 3.1(b) displays the detailed modular structure of a performance agent.
- ***Performance Library:*** The performance library is a well-defined class library that provides component developers and component users a standard component performance framework. This framework contains a set of well-defined performance tracking and analysis classes that support performance measurement of speed, throughput, resource utilization, availability, reliability, and scalability.

On each computing station over a network, there are two pre-defined performance tracking message queues. One is used to manage and control the performance tracking messages from different components to the performance library. And the other is used to manage and control the performance tracking messages from the performance library to a performance agent. The collected performance data is recorded in a repository at the server side.

4 Performance Tracking Techniques

Program tracking is one of important technique for program testing and performance evaluation [6]. Performance tracking is needed to support performance testing and evaluation. Its major purpose is to help engineers collect, track, and monitor component and system performance data. Two different performance-tracking techniques are being used. They are known as *time-based performance tracking*, and *volume-based performance tracking*. Both methods could be implemented manually or systematically. Here, we only highlight the general idea. The details can be found in the technical report [11].

A) Time-Based Performance Tracking

The focus of this method is to collect and track the component functional speed. The basic idea is to provide engineers with a set of pre-defined speed measurement functions as a part of the performance library. It enables engineers to track and analyze the processing speed from point A to point B on a program control flow inside software components. For instance, to monitor the execution time of a component's function, engineers can insert the time-based performance tracking code at the beginning and end point of a component function in a manual (or systematical) manner. The performance library computes the function execution time based on the collected speed performance traces. Typical applications of time-based performance tracking are listed below.

- Create a driver program by wrapping a component function call to check its execution speed.
- Check the execution time of a loop or a block of sequential statements in a component.
- Check the execution time of a functional transaction in components.
- Check the waiting-time for receiving a message or/and processing-time of received messages.

In addition, the tracking code can be used in any program code block, loops, repeatable logic, transaction sequences and functional scenarios in component-based programs. Based on time-based performance traces, the performance library computes the max/min/average processing time for a specific operation, function, or task in a component. The detailed description of this method and its implementation can be found in [11]. Figure 4.1(a) illustrates a simple example about how to add performance-tracking code to measure a component functional speed. As shown in the example, only three lines of sources are needed to insert into an original component function. Similarly, we can create a simple performance driver by wrapping a component function call to measure its execution speed.

B) Volume-Based Performance Tracking

The time-based performance tracking method is not suitable for stress and load testing due to its high system overhead because it generates performance traces whenever a component (or a program) exercised the inserted performance tracking code. To support performance measurement during load and stress testing, as well as component reliability and availability evaluation, the volume-based performance tracking method is developed. Unlike *time-based performance tracking*, where the focus is to monitor the processing time of component functions and operations, *volume-based performance tracking* focuses on measuring various throughputs in a system and its components. The typical examples are the throughputs of event/transaction/message processed in a component. This method is designed to evaluate component and system performance throughput, and support the measurement of component availability and reliability.

Its basic idea is to insert pre-defined program throughput tracking code into software components to collect and generate throughput performance traces based on a pre-defined time interval. During the execution of a component (or system), the volume-based performance tracking code counts the occurrence of a targeted event/transaction/message / operation in a component, and generates performance traces based on a specified time interval. The detailed descriptions about this method and its implementation could be found in [11]. Figure 4.1(b) illustrates a simple example to show how to add performance-tracking code for throughput measurement based on the given component performance library.

```
import PlibSpeed;
public class PerfSpeedComponent {
  public PerfSpeedComponent() {}
          :
  public void function() {
          :
  // Instantiate Speed class from library
  PlibSpeed ps = new PlibSpeed(PerformanceType,
  Component_id, User_Tag, Unit );

   ps.start(); // mark the start time

   // original function body

   ps.stop(); // mark the end time
  }
}
```

(a) Time-Based Tracking Example

```
import PlibThroughput;
public class PerfThroughputComponent {
  public PerfThroughputComponent() {}
          :
  public void function() {
          :
  // Instantiate Throughput class from library
  PlibThroughput pt = new PlibThroughput(
  PerformanceType, Component_id, User_Tag,
  Begin_Time, End_Time, Unit );

  while (current_time < end_time)
  {
    // original function body
    //  Library counts each call within test interval
    pt.event();
    // update current time
    current_time = new Date().getTime();
   }
  }
}
```

(b) Volume-Based Tracking Example

Fig 4.1: Sample Insertion of Performance Tracking Code

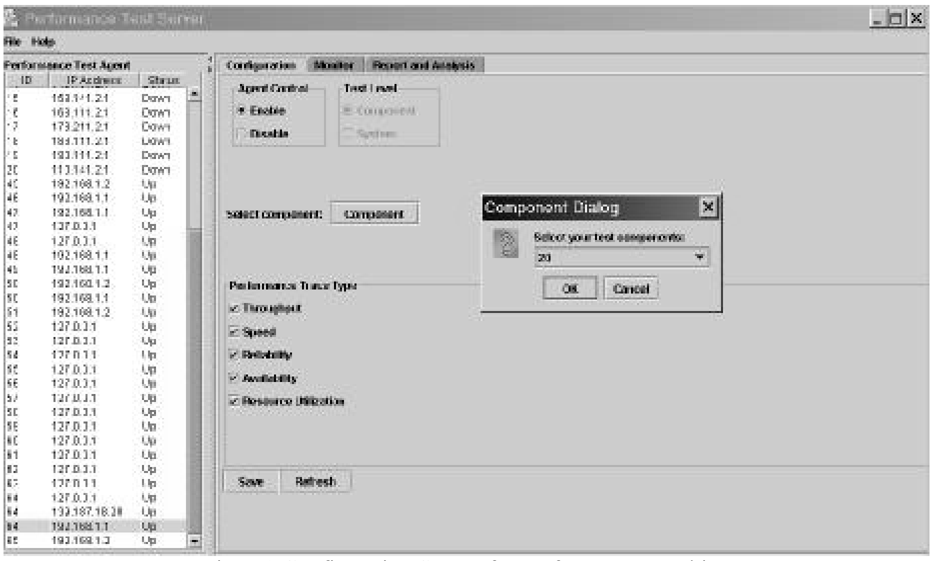
Performance testing engineers can use these methods to measure the occurrences of the incoming events (or messages) and outgoing events (or messages) of a component in a specific time slot. The performance data provides the information to compute the available-time and unavailable-time for component availability, and the uptime and downtime for component reliability. To support the transfer of component performance trace data, we have defined a consistent performance trace format to support six types of performance trace messages for component speed, throughput, reliability, availability, and system utilization respectively. Figure 4.2 shows their detailed formats. They have been used in the implementation to support the communications between the performance agents and the performance server in a distributed client-server environment. The detailed can be found in [11].

Agent ID	Node ID	Trace Type	Comp ID	User Tag	Begin Time	End Time	Trace Message

Fig. 4.2. Performance Trace Message Format

5 Application Examples and Case Study

Currently, we are working on a detailed case study to verify and validate this systematic solution and framework. Due to the limited space of this paper, we only report one application example and its related results in this section. A simple component-based application program, known as Stock Watcher, over a distributed environment, is selected as our experimental example. It was developed for the simulation of a stock watcher in which the stock watcher component is displaying stock price of a particular company. The stock watcher component is communicating with the stock server to update the stock price on the display. The application consists of two parts, a client and a stock server. Each includes a number of components.



Performance Test Agent

ID	IP Address	Status
15	163.141.2.1	Down
16	163.111.2.1	Down
17	173.211.2.1	Down
18	183.111.2.1	Down
19	193.111.2.1	Down
20	113.141.2.1	Down
45	192.168.1.2	Up
46	192.168.1.1	Up
47	192.168.1.1	Up
47	127.0.0.1	Up
48	127.0.0.1	Up
48	192.168.1.1	Up
49	192.168.1.1	Up
50	192.168.1.2	Up
50	192.168.1.1	Up
51	192.168.1.2	Up
52	127.0.0.1	Up
53	127.0.0.1	Up
54	127.0.0.1	Up
55	127.0.0.1	Up
56	127.0.0.1	Up
57	127.0.0.1	Up
58	127.0.0.1	Up
59	127.0.0.1	Up
60	127.0.0.1	Up
61	127.0.0.1	Up
62	127.0.0.1	Up
63	127.0.0.1	Up
64	127.0.0.1	Up
64	133.187.18.28	Up
64	192.168.1.1	Up
65	192.168.1.3	Up

Agent Control: [✓ Enable] Disable

Test Level: [✓ Component] System

Select component:

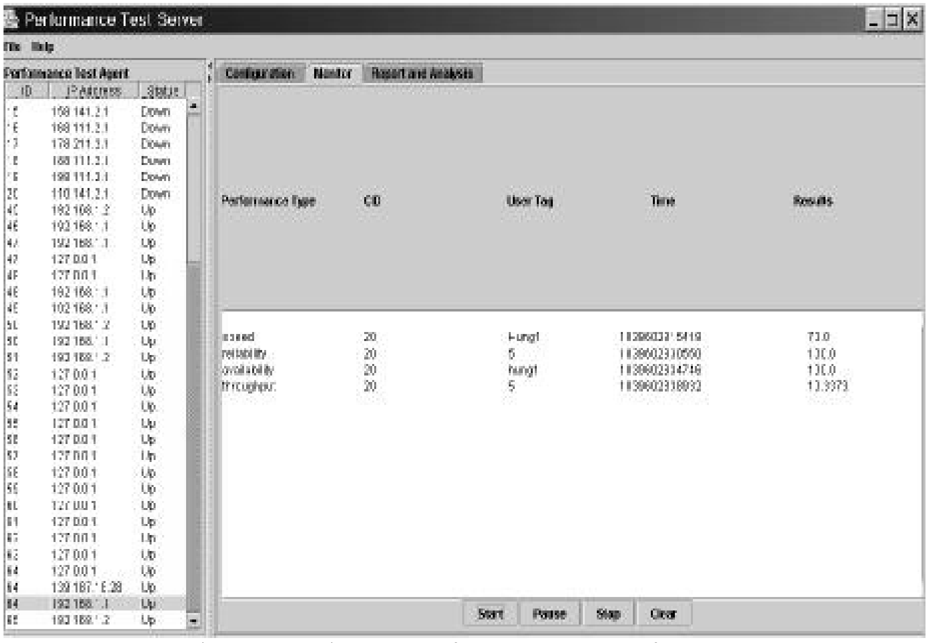
Component Dialog

Select your test components: 20

Performance Trace Type

✓ Throughput
✓ Speed
✓ Reliability
✓ Availability
✓ Resource Utilization

Fig. 5.1 Configuration Screen for Performance Tracking

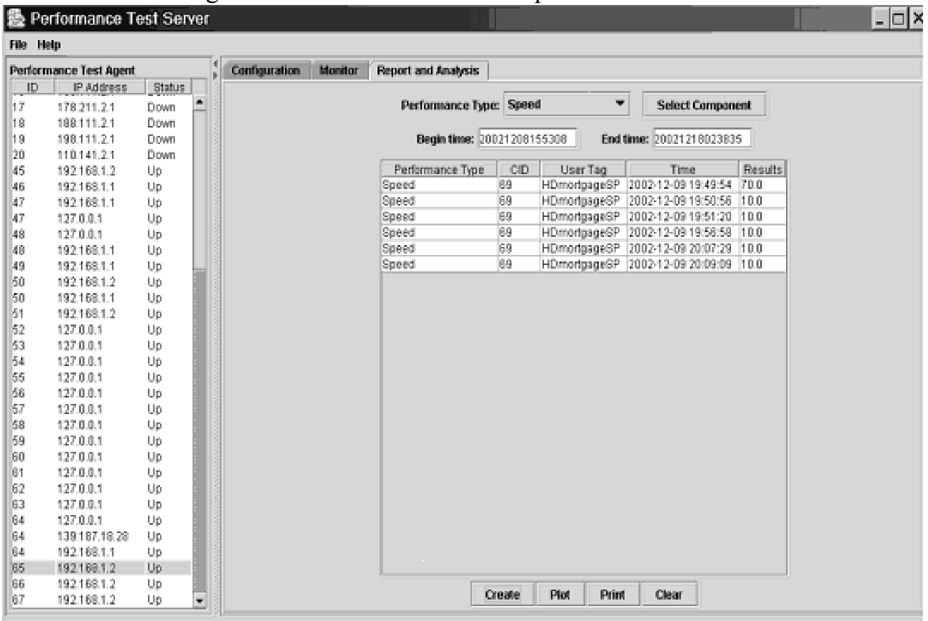


Performance Test Agent

ID	IP Address	Status
15	158.141.2.1	Down
16	168.111.2.1	Down
17	178.211.2.1	Down
18	188.111.2.1	Down
19	198.111.2.1	Down
20	110.141.2.1	Down
45	192.168.1.2	Up
46	192.168.1.1	Up
47	192.168.1.1	Up
47	127.0.0.1	Up
48	127.0.0.1	Up
48	192.168.1.1	Up
49	192.168.1.1	Up
50	192.168.1.2	Up
50	192.168.1.1	Up
51	192.168.1.2	Up
52	127.0.0.1	Up
53	127.0.0.1	Up
54	127.0.0.1	Up
55	127.0.0.1	Up
56	127.0.0.1	Up
57	127.0.0.1	Up
58	127.0.0.1	Up
59	127.0.0.1	Up
60	127.0.0.1	Up
61	127.0.0.1	Up
62	127.0.0.1	Up
63	127.0.0.1	Up
64	127.0.0.1	Up
64	139.187.18.28	Up
64	192.168.1.1	Up
65	192.168.1.2	Up

Performance Type	CID	User Tag	Time	Results
speed	20	hung1	1039603315419	70.0
reliability	20	5	1039602330590	100.0
availability	20	hung1	1039602334746	100.0
throughput	20	5	1039602338032	13.3373

Fig. 5.2: Monitor Screen for Component Performance



Performance Test Agent

ID	IP Address	Status
17	178.211.2.1	Down
18	188.111.2.1	Down
19	198.111.2.1	Down
20	110.141.2.1	Down
45	192.168.1.2	Up
46	192.168.1.1	Up
47	192.168.1.1	Up
47	127.0.0.1	Up
48	127.0.0.1	Up
48	192.168.1.1	Up
49	192.168.1.1	Up
50	192.168.1.2	Up
50	192.168.1.1	Up
51	192.168.1.2	Up
52	127.0.0.1	Up
53	127.0.0.1	Up
54	127.0.0.1	Up
55	127.0.0.1	Up
56	127.0.0.1	Up
57	127.0.0.1	Up
58	127.0.0.1	Up
59	127.0.0.1	Up
60	127.0.0.1	Up
61	127.0.0.1	Up
62	127.0.0.1	Up
63	127.0.0.1	Up
64	127.0.0.1	Up
64	139.187.18.28	Up
64	192.168.1.1	Up
65	192.168.1.2	Up
66	192.168.1.2	Up
67	192.168.1.2	Up

Performance Type: Speed

Begin time: 20021208155308 End time: 20021218023835

Performance Type	CID	User Tag	Time	Results
Speed	69	HDmortgageSP	2002-12-09 19:49:54	70.0
Speed	69	HDmortgageSP	2002-12-09 19:50:56	10.0
Speed	69	HDmortgageSP	2002-12-09 19:51:20	10.0
Speed	69	HDmortgageSP	2002-12-09 19:56:58	10.0
Speed	69	HDmortgageSP	2002-12-09 20:07:29	10.0
Speed	69	HDmortgageSP	2002-12-09 20:09:09	10.0

Fig. 5.3: Detailed Report Screen for Component Performance Results

Figure 5.1shows the GUI interface that allows engineers to configure performance-tracking features at the performance server. Figure 5.2 shows GUI interface supporting performance monitoring, and Figure 5.3 shows the GUI interface for performance reporting and analysis. Due to the limited scope of this paper, we only list three selected performance results based on our case study of the Stock Watcher program. Based on our application experience and case study, we found that the component performance framework and evaluation environment does provide a systematic solution to support performance evaluation of software components in a distributed environment. It has good potential to provide a systematic performance testing and evaluation for in-house built and third party components using a plug-in-and-measure approach. To achieve this goal, we have two tasks:

- Come out systematic solutions to construct measurable components. Now we create them manually in a systematic framework approach. Future, we will report how to construct measurable components later.
- Minimize and control the system overheads caused by performance tracking and evaluation during system execution.

Table 5.1 indicates that the performance-tracking overhead on CPU is very small and acceptable no matter the size of involved components. It also indicates that our current solution requires a minimal programming effort from engineers to work with the performance framework. We can insert the built-in tracking code either manually or systematically using a component wrapper approach.

Type	Avg PT w/o Tracking Code (milliseconds)	Avg PT with Tracking Code (milliseconds)	CPU Overhead (milliseconds)	Effort in inserting lines of tracking code
Speed	22	24	2	< 5
Throughput	10004.2	10004.4	0.2	5 to 10
Reliability	10004.6	10008.6	4	5 to 10
Availability	10036.6	10038	1.4	5 to 10
Resource Utilization	24.2	30	5.8	< 5

Table 5.1: System Overhead for Performance Tracking Code

6 Conclusions and Future Work

In this paper, we propose a systematic solution to measure component performance in a distributed environment based on a set of well-defined performance evaluation metrics and framework. Its major objective is to allow engineers use a standardized performance evaluation bed to measure component performance in a plug-in-and-measure approach. Its focus is to solve the addressed performance evaluation problems for components in component-based software development projects. In addition, we reported our design and implementation of a distributed component evaluation environment and framework, including its system architecture, framework, and related techniques in performance tracking and data collection. Based on our application experience, the solution is not only feasible and useful to support performance evaluation of homegrown components, but also demonstrate its potential value in supporting performance measurement of third-party components after converting them into measurable components. Currently, we are working on a number case study to evaluate and validate our solution and framework. We are also investigating new systematic solutions to measure the performance of component-based systems, and new methods to construct measurable components.

References

(1)	[CML00]
(2)	[Ch80]
(3)	[DBB96]
(4)	[Ga00]
(5)	[Ga01]
(6)	[GTW03]
(7)	[JW00]
(8)	[LK96]
(9)	[KM97]
(10)	[Va97]
(11)	[RH02]
(12)	[RP00]
(13)	[SS00]
(14)	[Ti95]

Bibliography

[CML00] Cahoon, B.; Mckinley, K.; Lu, Z.: Evaluating the Performance of Distributed Architectures for Information Retrieval Using a Variety of Workloads: ACM Transactions on Information Systems, January 2000; Vol. 18, No. 1, pp. 1-43.

[Ch80] Cheung, R.: A User-Oriented Software Reliability Model: IEEE Transitions on Software Engineering, March 1980, Vol. 6, No. 2, p. 118.

[DBB96] Dini, P.; Bochmann, G.; Boutaba, B.: Performance Evaluation for Distributed System Components: Proc. 2nd IEEE International Workshop on Systems Management, 1996.

[Ev99] Everett, W.: Software Component Reliability Analysis: IEEE Symposium on Application - Specific Systems and Software Engineering & Technology, 1999.

[Ga00] Gao, J.: Challenges and Problems in Testing Software Components: Proc. 3rd of ICSE2000's International Workshop on Component-based Software Engineering - Reflects and Practice, Limerick, Ireland, June 2000.

[Ga01] Gao, J.: Tracking Software Components: Journal of Object-Oriented Programming, August/September 2001.

[GTW03] Gao, J.; Tsao, J.; Wu, Y.: Testing and Quality Assurance for Component-Based Software. Artech House Inc., September 2003.

[JW00] Jogalekar, P.; Woodside, M.: Evaluating the Scalability of Distributed Systems: IEEE Transactions on Parallel and Distributed Systems, June 2000, Vol.11, No.6, pp.589-603.

[LK96] Laprie, J.; Kanoun, K.: Software Reliability and System Reliability - Handbook of Software Reliability Engineering. McGraw-Hill, New York, 1996.

[KM97] Krishnamurthy, S.; Mathur, A.: On the Estimation of Reliability of a Software System Using Reliabilities of its Components: IEEE The Ninth International Symposium on Software Reliability Engineering, Albuquerque, NM, November 02 - 05, 1997, p.146.

[Va97] Varsha, M.: Availability Analysis of Transaction Processing Systems based on User-Perceived Performance: IEEE 16th Symposium on Reliable Distributed Systems, Durham, NC, October 1997, p.10.

[RH02] Ravi, C.; Houng, D.: A Distributed Performance Measurement Framework for Software Components. Master Project Report, San Jose State University, Dec. 2002.

[RP00] Rudolf, A.; Pirker, R.: E-Business Testing: User Perceptions and Performance Issues: The First Asia-Pacific Conference on Quality Software, October 2000.

[SS00] Subraya, B.; Subrahmanya, S.: Object driven Performance Testing of Web Applications: The First Asia-Pacific Conference on Quality Software, October 2000.

[Ti95] Tian, J.: Integrating Time Domain and Input Domain Analyses of Software Reliability Using Tree-Based Models: IEEE Transactions on Software Engineering, December 1995, Vol. 21, No. 12, pp. 945-958.