The Performance Quantitative Model Based on the Specification and Relation of the Component

Baohua Wang, Danning Li, and Shun Zhang

1 School of Computer Science and Technology, Huaibei Normal University, Huaibei 235000, China
2 Guizhou Academy of Sciences, Guiyang 55001, China

Received: 5 June 2019; Accepted: 7 August 2019; Published: 11 August 2019

Abstract: The trustworthiness of software is crucial to some safety critical areas. Performance is an important attribute of software trustworthiness. Software component technology is the mainstream technology of software development. How to achieve the performance of component systems efficiently and accurately is a challenging issue for component-based software development. In this paper, the performance quantification method of the component is proposed. First, performance specification is formally defined. Second, a refinement relation is introduced and the performance quantification method of the component system is presented. Finally, a case study is given to illustrate the effectiveness of the method.

Keywords: performance; metric element; software component; trustworthiness

1. Introduction

The trustworthiness of software is a hot topic of research. Software performance is one of the attributes that affect software trustworthiness. With the expansion of software scale and the increase in software complexity, many software products encounter performance problems. Undoubtedly, many of these problems lead to lower software trustworthiness. In recent years, accidents caused by software performance problems have had a significant impact on our society. The third largest mobile phone company of Japan, Softbank, planned to attract users by reducing mobile fees. However, the computer system became paralyzed, and Softbank lost more than 100 million yen, as a result of the influx of a large number of users in a short time. Software in some safety-critical areas, such as aerospace control, finance, transportation and communication, need to achieve higher trustworthy degree [1,2].

Performance measurement is often a challenging task [3–6]. Automated assistance for software performance improvement has been described before based on measurements, performance models or both. To analyze measurements, the Paradyn tools [5] have extensive facilities for automated instrumentation and bottleneck searches for parallel applications, enhanced in [4] with historical data. IT flexibility has been posited as being a critical enabler in attaining competitive performance gains [7]. Other tools for the causal analysis of measurements include Poirot [8] and those in [9,10].

Performance is a crucial attribute of software systems [11]. When performance is measured early in the development process, the aim is to derive the appropriate architectural decisions that improve the performance of the system. Performance models are used to describe how system operations use resources and how resource contention affects operations [12]. Performance estimation at early stages of software development is difficult, as there are many aspects that may be unknown or uncertain, such as the design decisions, code, and execution environment. In many application domains, such as heterogeneous distributed systems, enterprise applications, and cloud computing, performance evaluation is also affected by external factors, such as increasingly fluctuating workloads.
and changing scenarios [13]. Wang and Casale used Bayesian inference and Gibbs sampling to model service demand from queue length data [14].

Software component technology is the mainstream technology of software development. How to obtain the performance of component systems efficiently and accurately is a challenging issue. This paper puts forward a performance quantification method of a component system. Component-Based Software Development (CBSD) has been an important area of research for almost three decades [15,16]. CBSD avoids duplication of effort, reduces the development cost and improves productivity [17,18]. Reusing the software component can reduce time and cost [19]. By reusing software components, CBSD avoids the repeated emergence of errors and improves the trustworthiness of software [20]. The software component has several definitions. A component is an opaque implementation of functionality, subject to third-party composition, and conformance with a component model (CMU/SEI definition) [21]. Szyperski’s definition is that the software component is a unit of composition with contractually specified interfaces and explicit context dependencies [22]. The authors of [23–25] adopted Szyperski’s definition. This paper also adopts Szyperski’s definition. 

One of the most critical processes in CBSD is the selection of a set of software components [26,27]. How to obtain the performance of component systems efficiently and accurately is a challenging issue for component system development. In this paper, the performance quantification method of the component is proposed. First, performance specification is formally defined. Second, a refinement relation is introduced, and the performance quantification method of the component system is presented. Finally, a case study is given to illustrate the effectiveness of the method.

The remainder of this paper is organized as follows. In Section 2, we introduce the performance and its metric elements. Section 3 proposes the refinement relationship. Section 4 presents the quantification method of the metric element and the computation model of performance. In Section 5, the performance computing method of the component system is presented. Section 6 gives a case study. Section 7 is the conclusion section.

2. Performance and Metric Elements

Performance is a non-functional feature of software [20]. It focuses on the timeliness displayed when it completes the function. The software system provides proper performance relative to the number of resources used under specified conditions. The object of performance measurement is called the metric element. The metric elements of the performance include the CPU utilization ratio, memory utilization ratio, disk I/O, number of concurrent users, throughput and average response time.

The CPU utilization ratio refers to the percentage of CPU time consumed by the user process and the system process. In a longer period of use, there is generally an acceptable upper limit, and many systems set the CPU utilization ratio to 85%.

The calculation of the memory utilization ratio is: \((1 - \text{free memory/total memory size}) \times 100\%\). The system requires at least a certain amount of free memory, which has an acceptable upper limit. Disk I/O is mainly used to measure the reading and writing ratio of the disk. It is measured by the percentage of unit time for reading and writing operations.

The maximum number of users is the number of users who simultaneously submit requests to the system at the same time.

Throughput is the number of unit time required of the system to complete users’ transactions. It reflects the system’s processing capabilities.

The average response time is the average time of transaction processing. The response time of the transaction is the time from submitting the request to completing it.

Component specification is a window for the component to transmit information [28]. Component specification not only provides an abstract definition of the internal structure of the component but also provides all the information needed for the component user to understand and use the component. Performance includes some metric elements. The trustworthiness of performance is directly calculated by metric elements.
3. The Refinement Relationship Based on the Specification

The performance is composed of metric elements. The trustworthiness of performance is calculated directly by metric elements. The specification of the performance’s metric element consists of four parts: name, checkpoint, flag and value.

**Definition 1.** (The specification of the metric element) A metric element consists of four parts: the name of the metric element, the checkpoint of the metric element, the flag of the metric element and the value of the metric element. It is denoted as a quadruple element: $= (\text{name}, \text{checkpoint}, \text{flag}, \text{value})$.

The flag of a metric element takes the values 1 and $-1$. The flag of a metric element represents the relation between the metric element and its value. When flag = 1, the larger the value of a metric element is, the better this metric element is. However, when flag = $-1$, the smaller the value of the metric element is, the better this metric element is. The flags of the metric elements are shown in Table 1.

<table>
<thead>
<tr>
<th>Metric Element Metric Element Abbreviation</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU utilization ratio</td>
<td>$\text{Cu}$</td>
</tr>
<tr>
<td>Memory utilization ratio</td>
<td>$\text{Mu}$</td>
</tr>
<tr>
<td>Disk I/O</td>
<td>$\text{Io}$</td>
</tr>
<tr>
<td>Maximum concurrent user number</td>
<td>$\text{Mn}$</td>
</tr>
<tr>
<td>Throughput</td>
<td>$\text{To}$</td>
</tr>
<tr>
<td>Average response time</td>
<td>$\text{Rt}$</td>
</tr>
</tbody>
</table>

**Definition 2.** (The specification of the Performance) A performance specification is a set consisting of $n$ metric elements. It is denoted as a set: $\text{per} = \{\text{element}_1, \ldots, \text{element}_i, \ldots, \text{element}_n\}$, where $\text{per}$ is the performance specification and $\text{element}_i$ is the specification of the metric element.

For example, a performance $\text{per}_1$ represents the throughput and the response time. Its description is that the throughput of the payment function is no less than 200 users per second when 100 users use the payment function, and the response time of the payment function is no more than 5 s when 200 users use the payment function. We use $\text{element}_1 = (\"\text{To}\", \"100 users\", 1, 20)$ to represent the throughput and $\text{element}_2 = (\"\text{Rt}\", \"200 users\", -1, 5)$ to represent the response time. Then, the specification of the performance is: $\text{per}_1 = \text{element}_1, \text{element}_2 = (\"\text{To}\", \"100 users\", 1, 20), (\"\text{Rt}\", \"200 users\", -1, 5)$.

After completing the specification of the metric element and performance, a refinement relation-based specification is introduced below.

**Definition 3.** (Refinement relation) For two given metric elements $\text{element}_1$ and $\text{element}_2$, we call $\text{element}_1$ the refinement of $\text{element}_2$, expressed as $\text{element}_1 \succ \text{element}_2$, if $(\text{element}_1.\text{name} = \text{element}_2.\text{name}) \land (\text{element}_1.\text{checkpoint} = \text{element}_2.\text{checkpoint}) \land (\text{element}_1.\text{flag} = \text{element}_2.\text{flag}) \land (((\text{element}_1.\text{flag} = 1) \land (\text{element}_1.\text{value} \geq \text{element}_2.\text{value})) \lor ((\text{element}_1.\text{flag} = -1) \land (\text{element}_1.\text{value} \leq \text{element}_2.\text{value})))$.

For example, if a metric element $\text{element}_1$ is ("To", "100 users", 1, 20) and a metric element $\text{element}_2$ is ("To", "100 users", 1, 18), then $\text{element}_1 \succ \text{element}_2$. If a metric element $\text{element}_3$ is ("Rt", "200 users", $-1$, 3) and a metric element $\text{element}_4$ is ("Rt", "200 users", $-1$, 5), then $\text{element}_3 \succ \text{element}_4$.

The refinement relationship has the following propositions:

**Proposition 1.** (Reflexivity) The refinement relation of the metric element satisfies the reflexivity, $\text{element}_i \succ \text{element}_i$. 

A metric element is satisfied with itself.

**Proposition 2.** (Antisymmetry) If $element_i \succ element_j$ and $element_j \succ element_i$, then $element_i = element_j$.

**Proof.** Because two cases of flag are discussed:

Case 1: When $(element_i, \text{flag} = 1)$, $element_i \succ element_j \Rightarrow (element_i, \text{name} = element_j, \text{name}) \land (element_i, \text{checkpoint} = element_j, \text{checkpoint}) \land (element_i, \text{flag} = element_j, \text{flag})$.

Because $element_i \succ element_j \Rightarrow element_i, \text{value} \geq element_j, \text{value}$ and $element_j \succ element_i \Rightarrow element_j, \text{value} \geq element_i, \text{value}$.

As a result, $element_i, \text{value} = element_j, \text{value}$.

Case 2: $(element_i, \text{flag} = -1)$

With the same proof of Case 1, $element_i, \text{value} = element_j, \text{value}$.

Therefore, $(element_i, \text{name} = element_j, \text{name}) \land (element_i, \text{checkpoint} = element_j, \text{checkpoint}) \land (element_i, \text{flag} = element_j, \text{flag}) \land (element_j, \text{value} = element_i, \text{value}) \Rightarrow (element_i = element_j)$.

As a result, $element_i = element_j$.

\[\square\]

**Proposition 3.** (Transitivity) If $element_i \succ element_j$ and $element_j \succ element_k$, then $element_i \succ element_k$.

**Proof.** Because two cases of flag are discussed:

Case 1: When $(element_i, \text{flag} = 1)$.

$element_i \succ element_j \Rightarrow (element_i, \text{name} = element_j, \text{name}) \land (element_i, \text{checkpoint} = element_j, \text{checkpoint}) \land (element_i, \text{flag} = element_j, \text{flag})$.

Because $element_i \succ element_j \Rightarrow (element_i, \text{name} = element_k, \text{name}) \land (element_i, \text{checkpoint} = element_k, \text{checkpoint}) \land (element_i, \text{flag} = element_k, \text{flag})$.

Because $(element_i, \text{name} = element_k, \text{name}) \land (element_i, \text{checkpoint} = element_k, \text{checkpoint}) \land (element_i, \text{flag} = element_k, \text{flag})$.

Two cases of flag are discussed:

Case 1: $(element_i, \text{flag} = 1)$

$element_i \succ element_j \Rightarrow (element_i, \text{value} \geq element_j, \text{value})$.

$element_j \succ element_k \Rightarrow (element_j, \text{value} \geq element_k, \text{value})$.

Thus, $(element_i, \text{value} \geq element_k, \text{value})$.

Case 2: $(element_i, \text{flag} = -1)$

This is similar to the proof of Case 1.

$(element_i, \text{value} \leq element_k, \text{value})$.

Therefore, $(element_i, \text{name} = element_k, \text{name}) \land (element_i, \text{checkpoint} = element_k, \text{checkpoint}) \land (element_i, \text{flag} = element_k, \text{flag}) \land ((element_i, \text{flag} = 1) \land (element_i, \text{value} \geq element_k, \text{value})) \lor ((element_i, \text{flag} = -1) \land (element_i, \text{value} \leq element_k, \text{value})))$.

Thus, $element_i \succ element_k$.

\[\square\]

For given two performance specifications $per_1$ and $per_2$, we call $per_1$ the refinement of $per_2$, expressed as $per_1 \succ per_2$, if $per_1, element_i \succ per_2, element_i$ for every $i$.

From these propositions, the refinement relation is a partial order. The relation of performance specifications is introduced below.
4. Quantification of Performance and the Metric Elements

In the following, a quantitative measurement method can help us choose the optimized performance. After quantifying the performance’s metric elements, the quantitative measurement of performance is computed.

**Definition 4.** (Quantitative measurement of the metric element) For the given specifications of two metric elements $element_1$ and $element_2$, we say that $element_2$ is refined $element_1$ in a quantitative way, donated by $element_1 \succ element_2$, calculated as follows. When $element_1.flag = 1$, $m(element_1, element_2) =$

\[
\begin{cases}
(element_1.value - element_2.value)^2, & element_1.value < element_2.value, \\
1, & element_1.value \geq element_2.value.
\end{cases}
\]

When $element_1.flag = -1$, $m(element_1, element_2) =$

\[
\begin{cases}
1, & element_1.value \leq element_2.value, \\
e^{-\frac{(element_2.value - element_1.value)}{element_2.value}}, & element_1.value > element_2.value.
\end{cases}
\]

The quantitative method has the following properties.

**Proposition 4.** (Boundness) Boundness is used to describe that $m(element_1, element_2)$ is in a certain range.

**Proof.** Since $m(element_1, element_2)$ is different depending on the flag, different cases are discussed.

Case 1: ($element_1.flag = 1$)

When $element_1.value < element_2.value$, it is obtained that $0 < \frac{(element_1.value - element_2.value)}{element_2.value}^2 < 1$.

When $element_1.value \geq element_2.value$, it is true that $m(element_1, element_2) = 1$.

As a result, $0 < m(element_1, element_2) \leq 1$.

Case 2: ($element_1.flag = -1$)

When $element_1.value \leq element_2.value$, $m(element_1, element_2) = 1$.

When $2 \times element_2.value > element_1.value > element_2.value \Rightarrow 0 < e^{-\frac{(element_2.value - element_1.value)}{element_2.value}} \leq 1$,

it is true that $0 < e^{-\frac{(element_2.value - element_1.value)}{element_2.value}} \leq 1$.

When $element_1.value \geq 2 \times element_2.value$, $m(element_1, element_2) = 0$.

As a result, $0 \leq m(element_1, element_2) \leq 1$.

In other words, $m(element_1, element_2)$ is a bounded function.

This proposition shows that a component’s metric element has a certain range in satisfying the target metric element.

\[\square\]

**Proposition 5.** (Keeping order) Keeping order means that $element_1 \succ element_2 \Rightarrow m(element_1, element_3) \geq m(element_2, element_3)$.

**Proof.** Two cases of the flag are discussed.

Case 1: ($element_1.flag = 1$)

Since $\frac{\partial m(element_1, element_3)}{\partial element_1.value} = 1 \times (element_1.value - element_2.value)^{-\frac{1}{2}} \geq 0$, $m(element_1, element_3)$ is an increasing function for $element_1$.

Because $element_1 \succ element_2$ can imply $element_1.value \geq element_2.value$, $m(element_1, element_3) \geq m(element_2, element_3)$.

As a result, if $element_1 \succ element_2$, then $m(element_1, element_3) \geq m(element_2, element_3)$.
Case 2: \((\text{element}_1, \text{flag} = -1)\)

Since \(\frac{\partial (\text{element}_1, \text{element}_2))}{\partial (\text{element}_2)} \leq 0\), \(m(\text{element}_1, \text{element}_2)\) is a decreasing function for \(\text{element}_1\).

Because \(\text{element}_1 \succ \text{element}_2\) implies \(\text{element}_1.\text{value} \leq \text{element}_2.\text{value}\), \(m(\text{element}_1, \text{element}_2) \geq m(\text{element}_2, \text{element}_3)\).

Then, \(\text{element}_1 \succ \text{element}_2\) implies \(m(\text{element}_1, \text{element}_3) \geq m(\text{element}_2, \text{element}_3)\).

\(\Box\)

**Proposition 6.** (Approximation) Approximation means that, when \(\text{element}_1\) is closing in a refinement way to \(\text{element}_2\), \(m(\text{element}_1, \text{element}_2)\) is increasing.

**Proof.** It is proved in the two cases of the flag of \(\text{element}_1\).

Case 1: \((\text{element}_1, \text{flag} = 1)\)

In this case, it is true that \(0 \leq \text{element}_1.\text{value} \leq \text{element}_2.\text{value}\).

Suppose \(x = \text{element}_2.\text{value} - \text{element}_1.\text{value}\), i.e., \(x\) denotes the distance from \(\text{element}_1.\text{value}\) to \(\text{element}_2.\text{value}\). then

\[
m(\text{element}_1, \text{element}_2) = \left(\frac{\text{element}_2.\text{value} - x}{\text{element}_2.\text{value}}\right)^{\frac{1}{2}}.
\]

\(0 \leq \text{element}_1.\text{value} \leq \text{element}_2.\text{value}\), we have

\[
\frac{\partial m(\text{element}_1, \text{element}_2)}{\partial x} = \left(-\frac{1}{2 \times \text{element}_2.\text{value}}\right) \times \left(\frac{\text{element}_2.\text{value} - x}{\text{element}_2.\text{value}}\right)^{-\frac{1}{2}} \leq 0.
\]

Case 2: \((\text{element}_1, \text{flag} = -1)\)

This case implies that \(\text{element}_2.\text{value} < \text{element}_1.\text{value}\). Suppose \(x = \text{element}_1.\text{value} - \text{element}_2.\text{value}\), i.e., \(x\) denotes the distance from \(\text{element}_2.\text{value}\) to \(\text{element}_1.\text{value}\). Then, \(m(\text{element}_1, \text{element}_2) = e^{\text{element}_2.\text{value}}\).

Since \(\text{element}_2.\text{value} < \text{element}_1.\text{value}\), \(\frac{\partial m(\text{element}_1, \text{element}_2)}{\partial x} = \left(-\frac{1}{\text{element}_2.\text{value}}\right) \times e^{\text{element}_2.\text{value}} \leq 0.

Both cases show that \(m(\text{element}_1, \text{element}_2)\) is a decreasing function over \(x\). Therefore, when \(\text{element}_1.\text{value}\) is closing to \(\text{element}_2.\text{value}\) in a refinement way, the degree of the metric element increases.

\(\Box\)

This proposition shows that the quantification is consistent with the refinement relation and that the quantitative model is order preserving.

**Definition 5.** (Quantitative measurement of component performance) Assuming that the component’s performance specification \(\text{per}_c\) and the target performance specification \(\text{per}_r\) have the same metric elements, we say that the component’s performance specification \(\text{per}_c\) satisfies the target performance specification \(\text{per}_r\) in a quantitative way, and it is represented as \(\text{per}_c \models_{\delta} \text{per}_r\), where \(\delta = (\delta_1 \times \delta_2 \times \cdots \times \delta_n)^{\frac{1}{2}}\) and \(\delta_1\) is the quantification of the \(i\)th metric element.

For example, there are the component’s performance specification \(\text{per}_c\) and the target performance specification \(\text{per}_r\). The component’s performance specification \(\text{per}_c\) is \([("\text{Rt}", "300 users", -1, 5)],\) and the target performance specification \(\text{per}_r\) is \([("\text{Rt}", "300 users", -1, 6)],\) It can be obtained that \(\text{per}_c \models_{\delta} \text{per}_r\), and \(\delta = 1\).

For another example, there are the component’s performance specification \(\text{per}_c\) and the target performance specification \(\text{per}_r\). The component’s performance specification \(\text{per}_c\) is \([("\text{Rt}", "200 users", -1, 5)], (\text{To}, 300 users, 1, 15)],\) and the target performance specification \(\text{per}_r\) is \([("\text{Rt}", "200 users", -1, 6)], (\text{To}, "300 users", 1, 20)].\) Through the calculation, we have \(\text{per}_c \models_{\delta} \text{per}_r\), and \(\delta = (\delta_1 \times \delta_2)^{\frac{1}{2}} = (1 \times \frac{10}{20})^{\frac{1}{2}} = 0.87.\)
The two target performance specifications are \{("R_t", "1000 users", 1, 6)\} and \{("R_t", "1000 users", 1, 6), ("R_o", "1000 users", 1, 20)\}. For each target performance specification, the performance specifications of the components are given to quantify, and the quantification is shown in Table 2. The examples illustrate the validity of quantitative method.

### Table 2. The performance quantification of Components.

<table>
<thead>
<tr>
<th>The Performance Specifications of Target Components</th>
<th>The Performance Specifications of Candidate Components</th>
<th>Quantitative (Value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{(&quot;pay-response&quot;, 1000 users&quot;, -1, 6)}</td>
<td>{(&quot;pay-response&quot;, &quot;1000 users&quot;, -1, 5)}</td>
<td>1</td>
</tr>
<tr>
<td>{(&quot;pay-response&quot;, &quot;1000 users&quot;, -1, 9)}</td>
<td>{(&quot;order-response&quot;, &quot;1000 users&quot;, -1, 4)}</td>
<td>0.71</td>
</tr>
<tr>
<td>{(&quot;pay-response&quot;, &quot;1000 users&quot;, -1, 6)}, {(&quot;pay-throughput&quot;, &quot;1000 users&quot;, 1, 20)}</td>
<td>{(&quot;pay-response&quot;, &quot;1000 users&quot;, -1, 6)}</td>
<td>0</td>
</tr>
<tr>
<td>{(&quot;pay-throughput&quot;, &quot;1000 users&quot;, 1, 30)}</td>
<td>{(&quot;pay-response&quot;, &quot;1000 users&quot;, -1, 6)}</td>
<td>0.87</td>
</tr>
</tbody>
</table>

5. Performance Computing of Component System

The graphical models are similar to Reliability Block Diagrams [29], but the computation models are different: the reliability of component system is calculated by series or parallel relation and the reliability of component; the performance of component system instead is calculated by the metric elements of component system, and the metric elements of component system are calculated by relation and the metric elements of component.

### 5.1. Series Relation

**Definition 6.** (Series relation) The series relation means that the component is executed in sequence, one component is executed after another component, and the post condition of the former is the precondition of the latter component. The series relation of the component is represented by "→".

Figure 1 depicts $C_1 \rightarrow C_2$. The CPU usage rate, memory utilization ratio, disk I/O, maximum user number, throughput and average response time of component serial relation system are calculated as shown in Table 3.

![Figure 1](image.png)

**Figure 1.** The series relation system.

### Table 3. The computation of the series relation system.

<table>
<thead>
<tr>
<th>Metric Element</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU utilization ratio</td>
<td>$C_{u_s} = \max(C_{u_1}, C_{u_2}, ..., C_{u_n})$</td>
</tr>
<tr>
<td>Memory utilization ratio</td>
<td>$M_{u_s} = \max(M_{u_1}, M_{u_2}, ..., M_{u_n})$</td>
</tr>
<tr>
<td>Disk I/O</td>
<td>$I_{o_s} = \max(I_{o_1}, I_{o_2}, ..., I_{o_n})$</td>
</tr>
<tr>
<td>Maximum concurrent user number</td>
<td>$M_{n_s} = \min(M_{n_1}, M_{n_2}, ..., M_{n_n})$</td>
</tr>
<tr>
<td>Throughput</td>
<td>$T_{o_s} = \min(T_{o_1}, T_{o_2}, ..., T_{o_n})$</td>
</tr>
<tr>
<td>Average response time</td>
<td>$R_{t_s} = \sum_{i=1}^{n} R_{t_i}$</td>
</tr>
</tbody>
</table>

$C_{u_i}$ is the CPU utilization rate of the $i$th component. $C_{u_s}$ is the CPU utilization rate of the component serial relation system. $M_{u_i}$ is the memory utilization of the $i$th component and $M_{u_s}$
is the memory utilization of the component serial relation system. $Io_i$ is the disk I/O of the $i$th component and $Io_s$ is the disk I/O of component serial relationship system. $Mn_i$ is the largest user number of the $i$th component, and $Mn_s$ is the largest number of users in a component serial relationship system. $To_i$ is the throughput of the $i$th component, and $To_s$ is the throughput of the component serial relationship system. $Rt_i$ is the average response time of the $i$th component, and $Rt_s$ is the average response time of the component serial relation system. $Max(Mu_1, Mu_2, ..., Mu_n)$ is a maximum function, and $Mn_s = min(Mn_1, Mn_2, ..., Mn_n)$ is a minimum function.

5.2. Parallel Relation

**Definition 7.** (Parallel relation) The parallel relation of the component refers to the simultaneous execution of the component and the equivalent of the component. The parallel relation structure can be supported by the successful execution of any of the components. The parallel relation is represented with $\Pi$.

For example, $C_1$ and $C_2$ are software components with a parallel relationship, denoted by $C_1 \Pi C_2$. The parallel relationship between $C_1$ and $C_2$ is shown in Figure 2. Metric elements of component parallel relation system are calculated as shown in Table 4.

![Figure 2. The parallel relation system.](image)

Table 4. The computation of the parallel relation system.

<table>
<thead>
<tr>
<th>Metric Element</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU utilization ratio</td>
<td>$Cu_s = max(Cu_1, Cu_2, ..., Cu_n)$</td>
</tr>
<tr>
<td>Memory utilization ratio</td>
<td>$Mu_s = max(Mu_1, Mu_2, ..., Mu_n)$</td>
</tr>
<tr>
<td>Disk I/O</td>
<td>$Io_s = max(Io_1, Io_2, ..., Io_n)$</td>
</tr>
<tr>
<td>Maximum concurrent user number</td>
<td>$Mn_s = min(Mn_1, Mn_2, ..., Mn_n)$</td>
</tr>
<tr>
<td>Throughput</td>
<td>$To_s = min(To_1, To_2, ..., To_n)$</td>
</tr>
<tr>
<td>Average response time</td>
<td>$Rt_s = max(Rt_1, Rt_2, ..., Rt_n)$</td>
</tr>
</tbody>
</table>

5.3. Choice Relation

**Definition 8.** (Choice relation) The choice relation means that one of the components is allowed to execute, but the only one component is allowed to perform at the same time. The preconditions of the components are the same in selection relationship. We use “+” to represent the choice relationship of the component.

For example, $C_1$ and $C_2$ are two components, and components $C_1$ and $C_2$ have a choice relationship. Figure 3 shows the choice relationship, which is shown as $C_1 + C_2$. Metric elements of component choice relation system are calculated as shown in Table 5.
Figure 3. The choice relation system.

Table 5. The computation of the choice relation system.

<table>
<thead>
<tr>
<th>Metric Element</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU utilization ratio</td>
<td>$C_{us} = \sum_{i=1}^{n} C_{ui} \times P_{i}$</td>
</tr>
<tr>
<td>Memory utilization ratio</td>
<td>$M_{us} = \sum_{i=1}^{n} M_{ui} \times P_{i}$</td>
</tr>
<tr>
<td>Disk I/O</td>
<td>$I_{os} = \sum_{i=1}^{n} I_{oi} \times P_{i}$</td>
</tr>
<tr>
<td>Maximum concurrent user number</td>
<td>$M_{ns} = \sum_{i=1}^{n} M_{ni} \times P_{i}$</td>
</tr>
<tr>
<td>Throughput</td>
<td>$T_{os} = \sum_{i=1}^{n} T_{oi} \times P_{i}$</td>
</tr>
<tr>
<td>Average response time</td>
<td>$R_{ts} = \sum_{i=1}^{n} R_{ti} \times P_{i}$</td>
</tr>
</tbody>
</table>

5.4. Concurrence Relation

Definition 9. (Concurrence relation) The concurrence relationship requires all components to execute at the same time, and the preconditions of concurrence relationship are the same. The concurrence relationship is represented with “∥”.

For example, $C_1$ and $C_2$ are two components, $C_1$ and $C_2$ have a concurrence relationship, and the two components should meet the same precondition $C_1.pre=C_2.pre$, and be remembered as $C_1∥C_2$. Figure 4 shows the concurrence relationship. Metric elements of component concurrence relation system are calculated as shown in Table 6.

Figure 4. The Concurrence relation system.
Table 6. The computation of the concurrence relationship system.

<table>
<thead>
<tr>
<th>Metric Element</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU utilization ratio</td>
<td>$Cu_s = \min (\sum_{i=1}^{n} Cu_i, 1)$</td>
</tr>
<tr>
<td>Memory utilization ratio</td>
<td>$Mu_s = \min (\sum_{i=1}^{n} Mu_i, 1)$</td>
</tr>
<tr>
<td>Disk I/O</td>
<td>$Io_s = \min (\sum_{i=1}^{n} Io_i, 1)$</td>
</tr>
<tr>
<td>Maximum concurrent user number</td>
<td>$Mn_s = \min (Mn_1, Mn_2, ..., Mn_n)$</td>
</tr>
<tr>
<td>Throughput</td>
<td>$To_s = \min (To_1, To_2, ..., To_n)$</td>
</tr>
<tr>
<td>Average response time</td>
<td>$Rt_s = \max (Rt_1, Rt_2, ..., Rt_n)$</td>
</tr>
</tbody>
</table>

5.5. Loop Relation

**Definition 10.** (Loop relation) The loop relation allows a component to execute $n$ times repeatedly. It is represented with $\odot n$.

For example, $C_1$ is a component that allows component $C_1$ to execute repeatedly $n$ times, and the loop relation is shown in Figure 5. Then, the component $C_1$ performs $n$ times loop as $C_1 \odot n$. The CPU utilization ratio, memory utilization ratio, disk I/O, maximum user number, throughput and average response time of loop relational system are calculated as shown in Table 7.

![Figure 5. The Loop relationship.](image)

Table 7. The computation of the loop relationship system.

<table>
<thead>
<tr>
<th>Metric Element</th>
<th>Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU utilization ratio</td>
<td>$Cu_s = Cu_1$</td>
</tr>
<tr>
<td>Memory utilization ratio</td>
<td>$Mu_s = Mu_1$</td>
</tr>
<tr>
<td>Disk I/O</td>
<td>$Io_s = Io_1$</td>
</tr>
<tr>
<td>Maximum concurrent user number</td>
<td>$Mn_s = Mn_1$</td>
</tr>
<tr>
<td>Throughput</td>
<td>$To_s = To_1$</td>
</tr>
<tr>
<td>Average response time</td>
<td>$Rt_s = n \times Rt_1$</td>
</tr>
</tbody>
</table>

6. An Illustrative Example

The following example shows the availability and effectiveness of the measurement model. Firstly, we give a logical structure of a component system, as shown in Figure 6.
The CPU utilization, memory utilization, disk I/O, maximum user number, throughput and average response time are shown in Table 8.

Table 8. The values of the metric elements of the component.

<table>
<thead>
<tr>
<th>Component</th>
<th>CPU Usage Rate</th>
<th>Memory Usage Rate</th>
<th>Disk I/O Usage Rate</th>
<th>Maximum Number of Users</th>
<th>Throughput</th>
<th>Average Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_1</td>
<td>0.4</td>
<td>0.5</td>
<td>0.4</td>
<td>500</td>
<td>300</td>
<td>3</td>
</tr>
<tr>
<td>C_2</td>
<td>0.2</td>
<td>0.3</td>
<td>0.1</td>
<td>600</td>
<td>350</td>
<td>4</td>
</tr>
<tr>
<td>C_3</td>
<td>0.2</td>
<td>0.2</td>
<td>0.3</td>
<td>450</td>
<td>200</td>
<td>5</td>
</tr>
<tr>
<td>C_4</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
<td>550</td>
<td>450</td>
<td>3</td>
</tr>
<tr>
<td>C_5</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>450</td>
<td>350</td>
<td>4</td>
</tr>
<tr>
<td>C_6</td>
<td>0.35</td>
<td>0.35</td>
<td>0.35</td>
<td>400</td>
<td>300</td>
<td>6</td>
</tr>
</tbody>
</table>

The calculation process is calculated according to the formula given above. The calculation of the metric element is shown as follows.

\[ C_{us} = \max(C_{u1}, \max(C_{u2}, C_{u3}), C_{u4}, C_{u5}, C_{u6}) = 0.4, \]
\[ M_{us} = \max(M_{u1}, \max(M_{u2}, M_{u3}), M_{u4}, M_{u5}, M_{u6}) = 0.5, \]
\[ I_{os} = \max(I_{o1}, \max(I_{o2}, I_{o3}), I_{o4}, I_{o5}, I_{o6}) = 0.4, \]
\[ M_{us} = \min(M_{u1}, \min(M_{u2}, M_{u3}), M_{u4}, M_{u5}, M_{u6}) = 400, \]
\[ T_{os} = \min(T_{o1}, \min(T_{o2}, T_{o3}), T_{o4}, T_{o5}, T_{o6}) = 300, \]
\[ R_{ts} = R_{t1} + \max(R_{t2}, R_{t3}) + R_{t4} + R_{t5} + R_{t6} = 21. \]

Table 9 is the computational results of performance metrics for component systems. Table 10 shows the target value of the metric element of the component system.

Table 9. The computational results of performance metrics for component system.

<table>
<thead>
<tr>
<th>Component</th>
<th>CPU Usage Rate</th>
<th>Memory Usage Rate</th>
<th>Disk I/O Usage Rate</th>
<th>Maximum Number of Users</th>
<th>Throughput</th>
<th>Average Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.4</td>
<td>0.5</td>
<td>0.4</td>
<td>400</td>
<td>300</td>
<td>21</td>
</tr>
</tbody>
</table>

Table 10. The target value of the metric elements.

<table>
<thead>
<tr>
<th>Component</th>
<th>CPU Usage Rate</th>
<th>Memory Usage Rate</th>
<th>Disk I/O Usage Rate</th>
<th>Maximum Number of Users</th>
<th>Throughput</th>
<th>Average Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.6</td>
<td>0.7</td>
<td>0.5</td>
<td>500</td>
<td>300</td>
<td>25</td>
</tr>
</tbody>
</table>

According to the computation, \( \delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \) and \( \delta_6 \) are 1, 1, 1, \( (\frac{400}{500})^{\frac{1}{3}} \), 1, and 1, respectively. Calculating the trustworthiness of the component system’s performance, \( \delta = (\delta_1 \times \delta_2 \times \cdots \times \delta_6)^{\frac{1}{6}} = (1 \times 1 \times 1 \times (\frac{400}{500})^{\frac{1}{3}} \times 1 \times 1)^{\frac{1}{6}} = 0.9635. \)

According to the computation, “maximum number of users” is the weak metric element of the component system. “Maximum number of users” of component \( C_6 \) is the minimum, thus component \( C_6 \) should be improved to obtain a better performance of the system.
The illustrative example is used to show the computational process of performance quantization. The weights of the metric elements are the same in the calculation; however, the weights could be different and determined according to the requirements of the component system.

7. Conclusions

Software component technology plays an important role in constructing software systems. The goal of this paper is to propose a quantification model of performance. The performance computing method of the component system is presented. Propositions for specification-based performance measurement are presented to illustrate the reasonableness of the method. An illustrative example is presented, showing that the proposed method is effective. Reliability is another key hot topic of the component system. Misra presents the methods to consider imperfect fault coverage and common-cause failures in the reliability analysis [30]. A computational model considering comprehensive reliability and performance is future work. The model will be extended to deal with reliability and security. We have the experience about the trustworthiness measurement of software in aerospace embedded software [1], and plan to evaluate the performance in a specific real case study about aerospace embedded software.

Author Contributions: B.W. gave the idea; B.W. and S.Z. did the experiments; D.L. interpreted the results; and B.W. wrote the paper

Funding: This work was supported by the National Key R&D Program of China (Grant No.2017YFC1601800, Name: Research and Application Demonstration of Information Technology in Food Safety Society); the International Cooperation Project Guizhou Academy of Sciences (Grant No.201901, Name: Security Assessment Model and Application Based on Food Safety Cloud Platform); the Natural Science Foundation of Anhui Province (Grant No.1708085MF159); and the Natural Science Project of Anhui Universities (Grant No. KJ2017A375).

Conflicts of Interest: The authors declare no conflict of interest.

References
4. Karavanic, K.L.; Miller, B.P. Improving Online Performance Diagnosis by the Use of Historical Performance Data, Acm/ieee Conference on Supercomputing. ACM 1999, 42. [CrossRef]


© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).