Research on the Utility Max-min Fair Algorithm of Resource Allocation

XU Tong ¹ and LIAO Jianxin ²

¹, ² State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications
296#, 10 Xitucheng Road, Haidian District, Beijing, China, 100876
{xutong, liaojianxin}@ebupt.com

Abstract: In this paper, the Utility Max-Min (UMM) fairness criteria is analyzed, and a
general algorithm of resource allocation based on UMM fairness is provided, named
system-scale, which can be applied in various problems of resource allocation in the field of
computer and communication. This algorithm supports the upper and lower bounds of
resource allocation; it also supports various utility functions that are strictly increasing and
continuous. The result of the algorithm is pareto-efficient and UMM fair. Because of the
avoidance of iterative procedure, system-scale is less complex than water-filling, a
well-known UMM fair algorithm. Moreover, the generalized system-scale can also act as a
utility min-max fair algorithm and be applied in environments such as load balancing and fair
job allocation.

Keywords: Utility, Max-min Fairness, Resource Allocation, Water-filling Algorithm,
System-scale Algorithm

1. INTRODUCTION

In the field of computer and communication, it’s a general problem to properly allocate
resource such as bandwidth, links, buffer storage, CPU time, balance etc. In this problem,
there are two main targets: efficiency and fairness. Most of the existing works concentrate on
efficiency. In recent years, research has been done on some scenarios of resource allocation
that take fairness for the main target [1]. Some fairness criterions were proposed, including the
max-min fairness [2], proportional fairness [3], harmonic mean fairness [4] etc. Among them,
max-min fairness was proved to be the fairest criteria [5,6], and was widely applied in the
allocation of bandwidth, wireless links and so on.

The aim of max-min fairness is to allocate “as much as possible to the poor users” [7]. It
has several variations: the original max-min fairness [8] supports neither the lower/upper
bounds nor the priority or weight of service; the generalized max-min (GMM) fairness [9] and
the general weighted (GW) fairness [10] improve the above two aspects respectively. However,
the common drawback of the above max-min fairness is that all of them support only the
resource for the criterion of fairness. When take the income of resource for the criterion, they

¹ This work is jointly supported by: (1) Specialized Research Fund for the Doctoral Program of Higher Education (No.
20030013006); (2) National Specialized R&D Project for the Product of Mobile Communications (Development and
Application of Next Generation Mobile Intelligent Network).
are invalid if the relationship between income and resource is not linear.

To solve this problem, the concept of “utility” in economics was involved. Literature [7,12~15] proposed the utility max-min (UMM) fairness, yet there is difference in the understanding of “utility”. In [12,15], utility represents levels of service quality satisfaction perceived by application, and the utility value has only relative significance. Thus it assumes that the utility functions of all the users have the same and limited range, e.g. [0,1]. This is a subjective concept. It cannot be extended to support objective criterions of fairness evaluation, such as resource, nor can it support priority or weight of service \[16\]. Therefore the UMM in [12,15] is a fairness criteria in narrow sense. In contrast, the utility in literature [7,13,14] has a general meaning. The utility function is regarded as levels of service quality or performance of application. So the range of utility functions of the users may be different. Such utility can unify both objective and subjective criterions of fairness, and support priority or weight of service as well. So the fairness criteria in [7,13,14] can be called the generalized UMM.

In UMM fair algorithms, the algorithms in [13,14] can be attributed to “water-filling” \[7\], which increase resource of all the users uniformly so that their utilities are always equal, until all the users are upper bounded or the system resource is used up. The complexity of water-filling is relatively high because of the iterative procedure. The algorithm in [12,15] is based on the narrow-sense UMM fairness, and requires that utility functions of all the users have the same range. This algorithm has comparatively lower complexity because of the avoidance of iterative procedure, yet the narrow meaning of UMM limits its application.

In this paper, we propose a general algorithm of resource allocation based on the generalized UMM \[7,13\], named system-scale. It inherits the ideology of the algorithm in [12,15], but overcomes its shortcomings. This algorithm can be applied in various problems of resource allocation especially in the field of computer and communication.

The content of this paper is organized as follows. Section 2 reviews the generalized UMM fairness criteria as well as the existence and uniqueness theorems. Section 3 describes the system-scale algorithm, and proves it to be accordant with the generalized UMM fairness. In section 4, the complexity of system-scale is analyzed and compared with that of water-filling. In section 5, the effectiveness of system-scale is testified in a basic model of UMM fair bandwidth allocation, and its efficiency is compared with that of water-filling briefly. The last section concludes the paper and gives prospect of system-scale.

2. THE GENERALIZED UMM FAIRNESS CRITERIA

The generalized utility “can be considered a measure of how much a user would be willing to pay for the resource” \[11\]. The generalized UMM fairness criteria in [7,13] is defined as follow (for simplicity, in the following part we just use “UMM” for the shortened form of “generalized UMM”):

Let \( S = \{1, 2, ..., N\} \) be the set of users, \( X \subset \Re^N \) be the possible resource allocation vector space, \( A_i \subset \Re \) be the possible resource allocation space of user \( i \), and \( u_i : A_i \rightarrow \Re \) be its utility function.

**Definition 1** Assume that \( u_i \) is continuous and strictly increasing for all \( i \), then a vector \( \bar{x} \) is “utility max-min (UMM) fair” on set \( X \) if and only if
\[
(\forall y \in X) (\exists i \in S) u_i(y_i) > u_i(x_i) \implies (\exists j \in S) u_j(y_j) < u_j(x_j) \leq u_i(x_i). 
\]
In other words, “a vector \( \bar{x} \) is UMM fair on set \( X \) if and only if increasing one component \( x_s \) must be at the expense of decreasing some other component \( x_t \) such that \( u_s(x_s) \leq u_s(x_s) \) in \([7,13]\). What to be mentioned here is the relationship between “UMM fair” and “pareto-efficient”. A vector of resource allocation is pareto-efficient if there is no way to reallocate resources such that one can increase its utility without decreasing the others” \([1]\). It’s easy to find that pareto-efficiency is the necessary but not sufficient condition of UMM fairness when all the utility functions are continuous and strictly increasing.

The existence theorem and uniqueness theorem of the UMM fair allocation is as follow. The detailed proofs can be found in \([7]\).

**Theorem 1 (Existence)** Assume that \( u_i \) is continuous and strictly increasing for all \( i \); if set \( X \) is convex and compact, then there exists a UMM fair vector on \( X \).

**Theorem 2 (Uniqueness)** If a UMM fair vector exists on a set \( X \), then it’s unique.

The UMM fairness defined above is a universal criteria. All the existing max-min fairness criteria can be explained as its special cases \([14]\). For example, UMM becomes the original max-min when \( u_i(x_i) = x_i \), and the GMM when \( u_i(x_i) = x_i - MR_i \) (where \( MR_i \) denotes the lower bound of resource for user \( i \)), and GW when \( u_i(x_i) = (x_i - MR_i)/w_i \) (where \( w_i \) denotes the weight of user \( i \)).

### 3. THE SYSTEM-SCALE ALGORITHM

Now we give a definition of system-scale, a general algorithm of resource allocation:

Let \( R \) be the amount of resource in the system, \( N \) be the number of users, \( S = \{1, 2, ..., N\} \) be the set of users, and \( PR_i \) and \( MR_i \) be the upper and lower bounds of resource for user \( i \), where \( MR_i \leq PR_i \). We define \( u_i(x) \) the utility function of user \( i \), which is strictly increasing and continuous on \([MR_i, PR_i]\). Then it’s obvious that on \([u_i(MR_i), u_i(PR_i)]\) there exists \( u_i^{-1}(x) \), the inverse function of \( u_i(x) \).

We first give the definitions of the extended function of inverse function, the generalized inverse function, and the system utility function:

**Definition 2** Let \( u(x) \) be a strictly increasing and continuous function on \([MR, PR]\), \( MR \leq PR \), \( u^{-1}(x) \) be the inverse function of \( u(x) \) on \([u(MR), u(PR)]\). If \( \alpha \leq u(MR) \leq u(PR) \leq \beta \), then the extended function of \( u^{-1}(x) \) on \([\alpha, \beta]\) is:

\[
eu^{-1}(x) = \begin{cases} 
MR & \alpha \leq x < u(MR) \\
u^{-1}(x) \cdot u(MR) & u(MR) \leq x \leq u(PR) \\
PR & u(PR) < x \leq \beta
\end{cases}
\]

It’s evident that \( u^{-1}(x) \) is strictly increasing and continuous, and \( eu^{-1}(x) \) is continuous and monotonously non-decreasing.

**Definition 3** Consider function \( f : A \to B \). Define \( f^{(-1)} : B \to A \) the generalized inverse
function of $f$, where $f^{(-1)}(x) = \inf\{y \in A : f(y) \geq x\}, \forall x \in B$.

**Definition 4** Let $eu_i^{-1}(x)$ be the extended function of $u_i^{-1}(x)$ on $[\min(u_i(MR_i) : i \in S), \max(u_i(PR_i) : i \in S)]$. Define $U(x)$ the system utility function if and only if $U^{(-1)}(x) = \sum_{i=1}^{N} eu_i^{-1}(x)$, where $U^{(-1)}(x)$ is the generalized inverse function of $U(x)$.

It's easy to prove that the domain of definition of $U(x)$ is $[\sum_{i=1}^{N} MR_i, \sum_{i=1}^{N} PR_i]$, and the range of $U(x)$ is $[\min(u_i(MR_i) : i \in S), \max(u_i(PR_i) : i \in S)]$, and $U(x)$ is increasing but may not be continuous.

In fact, the system utility function $U(x)$ is a useful tool in calculating the proper vector of resource allocation. It calculates the resource to be allocated for every user in the system so that the users’ utilities are as equal as possible. Since the resource of user $i$ is lower bounded and upper bounded by $MR_i$ and $PR_i$, sometimes users in the system cannot get the equal utility. For simplicity of calculation, definition 2 actually introduces the “extended utility”, which was defined as follow: when the utility of user $i$ is equal to its maximal utility $u_i(PR_i)$, its “extended utility” can be any value equal to or more than $u_i(PR_i)$; when the utility of user $i$ is equal to its minimal utility $u_i(MR_i)$, its “extended utility” can be any value equal to or less than $u_i(MR_i)$; when the utility of user $i$ is between $u_i(MR_i)$ and $u_i(PR_i)$, its “extended utility” is equal to its utility.

Therefore, all users in the system can get the equal “extended utility” $U(R)$ when the total resource in the system is $R$ ($\sum_{i=1}^{N} MR_i \leq R \leq \sum_{i=1}^{N} PR_i$).

Based on definition 4 we can give description of the system-scale (SS) algorithm:

0. Assume that before the start of the algorithm the system already know $u_i(x)$ and $u_i^{-1}(x)$ for all $i$.

1. if $R < \sum_{i=1}^{N} MR_i$ then {
2. run overload control process;
3. goto step 1;
4. }

4. else if $R = \sum_{i=1}^{N} MR_i$ then $\forall i \in S$ $x_i \leftarrow MR_i$;
5. else if $R \geq \sum_{i=1}^{N} PR_i$ then $\forall i \in S$ $x_i \leftarrow PR_i$;
6. else { if $\sum_{i=1}^{N} MR_i < R < \sum_{i=1}^{N} PR_i$
7. \( \forall i \in S \) calculate \( eu_i^{-1}(x) \) on \([\text{MIN}(u_i(MR_i): i \in S), \text{MAX}(u_i(PR_i): i \in S)]\);
8. resolve equation \( x = \sum_{i=1}^{N} eu_i^{-1}(U(x)) \) for \( U(x) \);
9. calculate \( U(R) \);
10. \( \forall i \in S \) if \( U(R) \geq u_i(PR_i) \) then \( x_i \leftarrow PR_i \);
11. else if \( U(R) \leq u_i(MR_i) \) then \( x_i \leftarrow MR_i \);
12. else \( x_i \leftarrow u_i^{-1}(U(R)) \);

The main idea of system-scale is: if the system resource cannot meet the minimum requirement of the users, then run overload control process and restart the algorithm; if the system resource just meet the minimum requirement of the users, then allocate the lower bound of resource for each user; if the system resource is equal to or more than the maximum requirement of the users, then allocate the upper bound of resource for each user; if the system resource is between the minimum and maximum requirements of the users, take \( \text{scale} \) of the system, and calculates the same “extended utility” that all users can get, then calculates the vector of resource allocation for every user according to definition 2.

The objective of overload control process in step 2 is to ensure the system resource not to be insufficient by stopping serving some users (i.e. deleting these users from \( S \)). Although the admission control process running beforehand can do the same work, still it cannot stop changes happened in resource or users, which will result in the insufficiency of system resource. Then in such case the overload control process is essential. The detailed discussion about admission control and overload control is beyond the range of this paper.

**Lemma 1** The vector of resource allocation generated by the system-scale algorithm is pareto-efficient.

Based on lemma 1, we can give theorem 3.

**Theorem 3** The vector of resource allocation generated by the system-scale algorithm is UMM fair.

Proofs of lemma 1 and theorem 3 are omitted due to lack of space.

4. COMPLEXITY ANALYSIS OF SYSTEM-SCALE

The complexity of system-scale is mainly dominated by its step 8, which is to resolve the equation \( x = \sum_{i=1}^{N} eu_i^{-1}(U(x)) \) for \( U(x) \). Let the complexity of step 8 be \( \text{O}(SE) \), and then the complexity of system-scale is \( \text{O}(SE) \) too.

For the sake of comparison, we now analyze the complexity of water-filling, another well-known UMM fair algorithm:

The algorithms in literature [13,14] can be attributed to water-filling. The water-filling algorithm in [14] can be described as follow.

0–6. the same process as step 1–6 in system-scale. the following process will run when
\[
\sum_{i=1}^{N} MR_i < R < \sum_{i=1}^{N} PR_i.
\]

7. \( \forall i \in S \ x_i \leftarrow MR_i \);

8. sort \( \{u_i(MR_i)\} \), assume the result be \( u^{(1)} < u^{(2)} < \cdots < u^{(W)} \); let \( u^{(W+1)} = \infty \); let \( S^{(1)} \), \( S^{(2)} \), \ldots, \( S^{(W)} \) respectively denote the set corresponding to \( u^{(1)} \), \( u^{(2)} \), \ldots, \( u^{(W)} \);

9. let \( R1 = R - \sum_{i \in S} MR_i \); \( S1^{(1)} = \emptyset \); //initiation of “for” loop, where \( R1 \), \( S1^{(i)} \) be the temporary variables

10. for ( \( v = 1; \ R1 > 0 \ &\ & v \leq W; \ v[++]\) }

11. \( S1^{(v)} \leftarrow S1^{(v)} \cup S^{(v)} \);

12. \( R1 \leftarrow R1 + \sum_{i \in S^{(v)}} MR_i \);

13. let \( R2 = R1 \); \( S2^{(0)} = \emptyset \); \( S2^{(1)} = S1^{(v)} \); \( m = 1 \); //initiation of “while” loop, where \( R2 \), \( S2^{(m)} \), \( m \) be the temporary variables

14. while ( \( S2^{(m)} \neq \emptyset \) ) and ( \( S2^{(m)} = S2^{(m-1)} \) ) do {

15. resolve the equation \( x = \sum_{i \in S^{(m)}} u_i^{-1}(U^{(m)}(x)) \) for \( U^{(m)}(x) \); //\( U^{(i)}(x) \) be the temporary variables of function

16. calculate \( U^{(m)}(R2) \);

17. \( \forall i \in S^{(m)} \ x_i \leftarrow u_i^{-1}(U^{(m)}(R2)) \);

18. \( S2^{(m+1)} \leftarrow S2^{(m)} - \{i \mid x_i > PR_i \| u_i(x_i) > u_i^{-1}, \forall i \in S^{(m)}\} \);

19. \( \forall i \in S^{(m)} \) if \( x_i > PR_i \) then \( x_i \leftarrow PR_i \);

20. \( \forall i \in S^{(m)} \) if \( u_i(x_i) > u_i^{-1} \) then \( x_i \leftarrow u_i^{-1}(u_i^{-1}) \);

21. \( R2 \leftarrow R2 - \sum_{i \in S^{(m+1)} - S^{(m)}} x_i \);

22. \( m \leftarrow m + 1 \); //while

23. if \( S2^{(m)} = \emptyset \) then { \}

24. \( S1^{(v+1)} \leftarrow S1^{(v)} - \{i \mid x_i = PR_i, \forall i \in S1^{(v)}\} \);

25. \( R1 \leftarrow R1 - \sum_{i \in S1^{(v)} - S1^{(v+1)}} PR_i \); //if

26. else \( v \leftarrow W + 1 \); //terminate “for” loop //for

The main idea of water-filling is: if the system resource is less than or equal to the minimum requirement of the users, or more than or equal to the maximum requirement of the users, the process is just the same as system-scale; if the system resource is between the minimum and maximum requirements of the users, calculate all users’ inverse functions, and allocate as much as \( MR_i \) of resource for user \( i \); classify and sort the users by utility;
allocate the remaining resource to the users in the set with the lowest utility, until its resource
reaches its $PR$, or its utility reaches the next higher class of utility; repeat “for” loop until
resource is insufficient, or all users have resource of its $PR$, or have utility of its highest
utility.

The complexity of water-filling is mainly dominated by its step 15, which is to resolve
the equation $x = \sum_{i \in S^{(m)}} u_i^{-1}(U^{(m)}(x))$ for $U^{(m)}(x)$. Its complexity can be considered to be
the same as that of step 8 of system-scale, which is O(SE). To water-filling, the worst case is
that: (1) $u_i(MR)$ are different from each other for all i; (2) $\forall j \in S^{(i)}$, $i=1,2,..,W-1$,$u_j(PR_j) \in (u_i^{(i+1)},u_i^{(i+2)})$; (3) $R \in (\sum_{i \in S^{(w)}} PR_i,\sum_{i \in S} PR_j)$. Then because of (1), the “for”
loop must run $W = N$ times; because of (2) and (3), the “while” loop in the first “for” loop
will run 1 time, and in the Nth “for” loop it will run 3 times, and in the middle N-2 times
“for” loops it’ll run at most 2 times for each loop. That is, in the worst time step 15 will run
1+3+2*(N-2)=2N times, so we can say that the complexity of water-filling is O(N*O(SE)).

Since it’s very difficult to resolve equation with the original form of functions, in reality,
functions can be approximated by the continuous piecewise linear shape [12,15], then
O(SE)=O(N*K) [12], where K denotes the number of the subrange and is proportional to the
accuracy of approximation. Then the complexity of system-scale is O(N*K), while that of
water-filling is O(N^2*K).

From the above we can see that the complexity of system-scale is much lower than that of
water-filling. The reason lies in that: water-filling probes and searches the different upper
bounds of the users with iterative procedure, so it’ll resolve the equation in each iteration;
while system-scale includes the different upper bounds of the users into their extended
functions of the inverse function of utility function, then it avoids iterative procedure and
resolve equation only once.

5. SIMULATION

In this section we testify the effectiveness as well as the efficiency of system-scale in a
basic model of UMM fair bandwidth allocation. Consider a communication system consisting
of four end-to-end applications s1, s2, s3 and s4. The bandwidth resource of the system is R
and is shared by the four applications. Table 1 gives parameters of the applications, including
the utility functions and the lower/upper bounds of resource and utility. Figure 1(a) illustrates
the utility functions.
Table 1
Parameters of the simulation model

<table>
<thead>
<tr>
<th>Application</th>
<th>( u_i(x) )</th>
<th>MR(_i)</th>
<th>PR(_i)</th>
<th>( u_i(MR(_i)) )</th>
<th>( u_i(PR(_i)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>( u_1(x) = x )</td>
<td>4</td>
<td>20</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>s2</td>
<td>( u_2(x) = 6x^{1/3} )</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>15.72</td>
</tr>
<tr>
<td>s3</td>
<td>( u_3(x) = 9.28 + 7\times\arctan((x - 8)/2) )</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>18.56</td>
</tr>
<tr>
<td>s4</td>
<td>( u_4(x) = 8 )</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

*Here \( u_i(x) \) is only utility function on \([MR\(_i\), PR\(_i\)]\); \( \forall i \in S \) \( u_i(x) = 0 \) if \( x \in [0, MR\(_i\)] \), and \( u_i(x) = u_i(PR\(_i\)) \) if \( x \in (PR\(_i\), +\infty) \).

Figure 1
Process of system-scale on UMM fair bandwidth allocation

In the above model, we use some classical utility functions: s1 is a linear shape application with lower/upper bounds of bandwidth resource, which means its utility is proportional to the bandwidth resource it gets; s2 is a typical elastic application\(^{[11]}\), which is characterized by the strictly concave utility function with no lower bound; s3 is something like the rate-adaptive/delay-adaptive application\(^{[11]}\), whose utility function is convex but not concave in a neighborhood around zero; s4 is a hard real-time application\(^{[11]}\), whose lower bound and upper bound of bandwidth resource are equal.

Figure 1(b/c/d) illustrate the process of calculating the system utility function \( U(x) \), where (b) shows the extended functions of the inverse functions of the utility functions on \([MIN(u_i(MR\(_i\))), MAX(u_i(PR\(_i\)))]\) (i.e.\([0, 20]\)); (c) shows the sum of the above extended
functions; and (d) is the system utility function \( U(x) \) calculated according to definition 4, whose domain of definition is \( \left[ \sum_{i=1}^{4} MR_i, \sum_{i=1}^{4} PR_i \right] \) (i.e. [10, 60]) and range is \( [MIN(u_i(MR_i)), MAX(u_i(PR_i))] \) (i.e. [0, 20]).

Resource \( R \) be a variable, the result of system-scale is given in table 2.

### Table 2

| \( R \) | \( U(R) \) | Resource vector \( \hat{x}=(x_1, x_2, x_3, x_4) \) | Utility vector \( \hat{u}=(u_1(x_1), u_2(x_2), u_3(x_3), u_4(x_4)) \) | Status vector \( \text{AS}=(AS_1, AS_2, AS_3, AS_4)* \)
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_0=10 )</td>
<td>0</td>
<td>(4, 0, 0, 6)</td>
<td>(4, 0, 0, 8)</td>
<td>(M, M, M, MP)</td>
</tr>
<tr>
<td>( R_1=15 )</td>
<td>2.338</td>
<td>(4, 0.059, 4.941, 6)</td>
<td>(4, 2.338, 2.338, 8)</td>
<td>(M, O, O, MP)</td>
</tr>
<tr>
<td>( R_2=35 )</td>
<td>12.05</td>
<td>(12.05, 8.11, 8.84, 6)</td>
<td>(12.05, 12.05, 12.05, 8)</td>
<td>(O, O, O, MP)</td>
</tr>
<tr>
<td>( R_3=54 )</td>
<td>17.40</td>
<td>(17.40, 18, 12.60, 6)</td>
<td>(17.40, 15.72, 17.40, 8)</td>
<td>(O, P, O, MP)</td>
</tr>
<tr>
<td>( R_4=59 )</td>
<td>19</td>
<td>(19, 18, 16, 6)</td>
<td>(19, 15.72, 18.56, 8)</td>
<td>(O, P, P, MP)</td>
</tr>
<tr>
<td>( R_5=65 )</td>
<td>/</td>
<td>(20, 18, 16, 6)</td>
<td>(20, 15.72, 18.56, 8)</td>
<td>(P, P, P, MP)</td>
</tr>
</tbody>
</table>

* \( AS_i \) denotes the relationship among \( x_i \), \( MR_i \) and \( PR_i \): \( AS_i=M \) if \( x_i=MR_i \neq PR_i \); \( AS_i=O \) if \( x_i \in (MR_i, PR_i) \); \( AS_i=P \) if \( x_i=PR_i \neq MR_i \); and \( AS_i=MP \) if \( x_i=MR_i=PR_i \).

According to table 2, the utility of each user is equal except for those limited by the lower/upper bounds. Furthermore, when the system resource is not superfluous (i.e. \( R<\sum PR_i=60 \), correspond to \( R_0\sim R_4 \)), the sum of the resource allocated for the users (\( \sum x_i \)) is equal to the system resource (\( R \)), which means that the allocation is necessary. On the other hand, the resource each user get (\( x_i \)) is no more than its upper bound (\( PR_i \)), which means that the allocation is sufficient. When the system resource is superfluous (i.e. \( R>\sum PR_i=60 \), correspond to \( R_5 \)), \( x_i \) is equal to its \( PR_i \) and \( \sum x_i \) is equal to \( \sum PR_i \), which mean the allocation is both sufficient and necessary. Then we can conclude that the result in table 2 is UMM fair.

Both system-scale and water-filling can work out the same result in the above model. The equation-resolution step of system-scale (i.e. step 8) always run one time regardless of the value of resource \( R \), while the counterpart of water-filling (i.e. step 15) run different times according to the value of \( R \): 1 time for \( R_0, R_1 R_2 \) and \( R_5 \), 2 times for \( R_3 \) and 3 times for \( R_4 \). This indicates that system-scale is more efficient and less complex than water-filling.

### 6. CONCLUSION

UMM is a fairness criterion for resource allocation that is widely applied in the field of computer and communication. Most of the UMM fair algorithms are attributed to water-filling, whose complexity is quite high because of the iterative procedure. In this paper, we propose a new UMM fair algorithm: system-scale, which can be used in various problems of resource allocation. This algorithm supports the lower/upper bounds of resource allocation; it also supports various utility functions that are strictly increasing and continuous. Because of the avoidance of iterative procedure, the complexity of system-scale is much lower than that of water-filling. Simulation testified the effectiveness and efficiency of system-scale.

Moreover, literature [7] point out that if \( \hat{x} \) be the UMM fair vector on \( X \), then \( \hat{x} \) be
the utility min-max fair vector on $-X$. This indicates that system-scale is also a utility min-max fair algorithm, which can be applied in problems such as load balance, job allocation etc. Therefore, system-scale is a general-purpose fair algorithm with good prospect.

REFERENCES