GAME THEORY BASED LOAD BALANCED JOB ALLOCATION

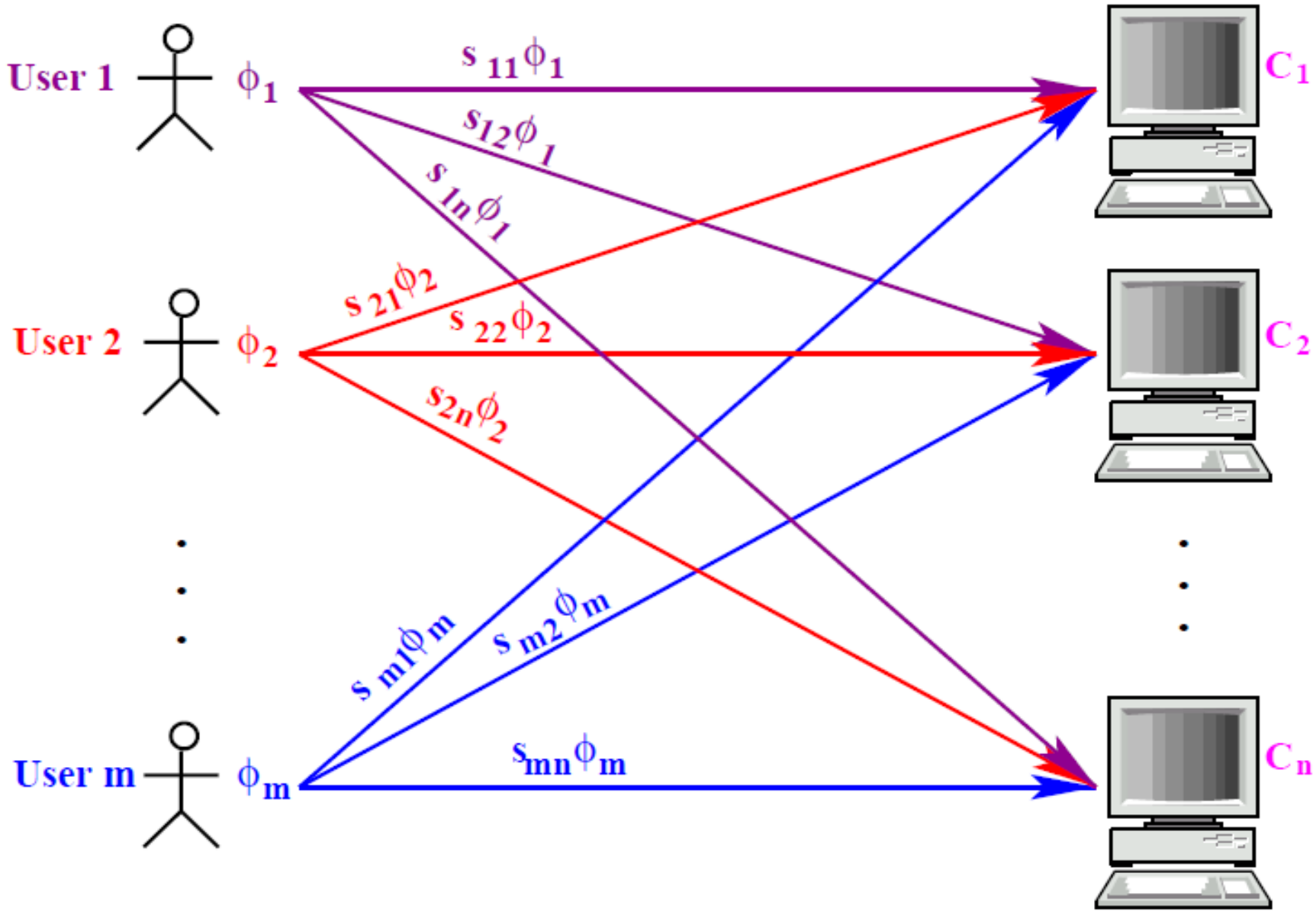
Load balancing as a non-cooperative game among users

We consider a ***distributed system*** that consists of *n* heterogeneous computers shared by *m* users.

Each computer is modeled as an M/M/1 queueing system (i.e. Poisson arrivals and exponentially distributed processing times) and is characterized by its average processing rate $\mu_i, i = 1, \dots, n$.

Jobs are generated by user *j* with an average rate ϕ_j, and
$\Phi = \Sigma_{j=1}^{m} \phi_j$ is the total job arrival rate in the system.

The total job arrival rate must be less than the aggregate processing rate of the system (i.e. $\Phi \leq \Sigma_{i=1}^{n} \mu_i$).


User 1
ϕ_1
User 2
ϕ_2
User m
ϕ_m
$s_{11}\phi_1$
$s_{12}\phi_1$
$s_{1n}\phi_1$
$s_{21}\phi_2$
$s_{22}\phi_2$
$s_{2n}\phi_2$
$s_{m1}\phi_m$
$s_{m2}\phi_m$
$s_{mn}\phi_m$
C_1
C_2
C_n

The problem faced by users is to decide on how to distribute their jobs to computers such that they will operate optimally.

Thus, user $j, (j = 1, \ldots, m)$ must find the fraction s_{ji} of its workload that is assigned to computer i ($\Sigma_{i=1}^{n} s_{ji} = 1$ and $0 \leq s_{ji} \leq 1, i = 1 \ldots n$) such that the ***expected execution time*** of its jobs is ***minimized***.
Once s_{ji} are determined, user j sends jobs to computer i at a rate given by $s_{ji}\phi_j$ (in jobs/s).

The problem is formulated as a ***non-cooperative game among users*** under the assumption that users are 'selfish', i.e. they minimize the expected response time of their own jobs.

Recall that s_{ji} is the fraction of workload that user j sends to computer i.
The vector $\mathbf{s}_j = (s_{j1}, s_{j2}, \ldots, s_{jn})$ is the ***load balancing strategy*** of user j, $j = 1, \ldots, m$.
The vector $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m)$ is the ***strategy profile*** of the load balancing game.

Since each computer is modeled as an M/M/1 queueing system,
the expected response time at computer i is given by

$$F_i(\mathbf{s}) = \frac{1}{\mu_i - \sum_{j=1}^{m} s_{ji}\phi_j}$$

Thus, the overall expected response time of user j is given by

$$D_j(\mathbf{s}) = \sum_{i=1}^{n} s_{ji} F_i(\mathbf{s}) = \sum_{i=1}^{n} \frac{s_{ji}}{\mu_i - \sum_{k=1}^{m} s_{ki}\phi_k}$$

The goal of user j is to find a feasible load balancing strategy $\boldsymbol{s}_j$ such that $D_j(\mathbf{s})$ is minimized.
The decision of user j depends on the load balancing decisions of the other users since $D_j(\mathbf{s})$ is a function of $\mathbf{s}$.

Feasible strategy profile.

A *feasible load balancing strategy profile* is a strategy profile $\mathbf{s}$ that satisfies the following restrictions:

(i) Positivity: $s_{ji} \geq 0,\ i = 1, \ldots, n,\ j = 1, \ldots, m;$

(ii) Conservation: $\sum_{i=1}^{n} s_{ji} = 1,\ \ j = 1, \ldots, m;$

(iii) Stability: $\sum_{j=1}^{m} s_{ji}\,\phi_j < \mu_i,\ \ i = 1, \ldots, n.$

Noncooperative load balancing game.

The *Noncooperative load balancing game* consists of a set of players, a set of strategies, and preferences over the set of strategy profiles:

(i) Players: The m users.

(ii) Strategies: Each user's set of feasible load balancing strategies.

(iii) Preferences: Each user's preferences are represented by its expected response time (D_j). User j prefers strategy profile $\mathbf{s}$ to strategy profile $\mathbf{s}'$ if and only if $D_j(\mathbf{s}) < D_j(\mathbf{s}')$.

In order to obtain a load balancing scheme for the distributed system we need to solve the above game. The most commonly used solution concept for such games is that of Nash equilibrium.

Nash equilibrium.

A *Nash equilibrium* of the load balancing game defined above is a strategy profile $\mathbf{s}$ such that for every user j $(j = 1, \ldots, m)$:

$$\mathbf{s}_j \in \underset{\tilde{\mathbf{s}}_j}{\operatorname{argmin}} D_j(\mathbf{s}_1, \ldots, \tilde{\mathbf{s}}_j, \ldots, \mathbf{s}_m).$$

Nash equilibrium for our load balancing game is a strategy profile with the property that no user can decrease its expected job execution time by choosing a different load balancing strategy given the other user's load balancing strategies.

In other words a strategy profile $\mathbf{s}$ is a Nash equilibrium if no user can benefit by deviating unilaterally from its load balancing strategy to another feasible one.

For our load balancing game there exists a ***unique*** Nash equilibrium because the expected response time functions are continuous, convex and increasing.

Remark. In general Nash equilibria are defined in terms of mixed strategies which are probability distributions over the set of pure strategies.
Here, we are interested only in ***pure strategy*** equilibria.
In pure strategy equilibria a user (player) chooses a unique strategy from the set of available strategies, whereas in mixed strategy equilibria he chooses a probability distribution over the set of strategies available to him.

An alternative definition of the Nash equilibrium will be used to determine a solution for our load balancing game:
Nash equilibrium can be defined as a strategy profile for which every user's load balancing strategy is a ***best reply*** to the other users strategies.
This best reply for a user will provide a minimum expected response time for that user's jobs given the other users' strategies.

This definition provides a method to determine the ***structure of the Nash equilibrium*** for our load balancing game.

A divide-and-conquer approach is employed:

1. Determine the best reply strategies $\mathbf{s}_j$ for each user j,
2. Find a strategy profile $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m)$ for which $\mathbf{s}_j$ is the best reply of user j, for $j = 1,2, \ldots, m$.

The best reply of user j, $j = 1,2, \ldots, m$, is the strategy profile that obtains the minimum expected response time for user j jobs with respect to the other users strategies.

Let $\mu_i^j = \mu_i - \Sigma_{k=1,k\neq j}^{m} s_{ki}\phi_k$ be the *available processing* rate at processor i as seen by user j.

The problem of computing the best reply strategy of user j $(j = 1, \ldots, m)$ reduces to ***computing the optimal strategy*** for a system with one user and n computers having μ_i^j $(i = 1, \ldots, n)$ as processing rates and ϕ_j as the user's job arrival rate in the system.

Computing the optimal strategy for a system with one user j and n computers having as processing rates the *available processing* rate as seen by user j can be expressed as the optimization problem $\text{BEST}-\text{REPLY}_j$:

$$\min_{\mathbf{s}_j} D_j(\mathbf{s})$$

subject to the constraints

$$s_{ji} \geq 0, \qquad i = 1, \ldots, n,$$
$$\sum_{i=1}^{n} s_{ji} = 1,$$
$$\sum_{k=1}^{m} s_{ki}\, \phi_k < \mu_i, \quad i = 1, \ldots, n.$$

Note that the strategies of all the other users are kept fixed, thus the variables involved in $\text{BEST}-\text{REPLY}_j$ are the load fractions of user j, i.e. $\mathbf{s}_j = (s_{j1}, s_{j2}, \ldots, s_{jn})$.

First, the best reply strategy of each user will be characterized, then an algorithm for computing this strategy considering our model will be developed.

The best reply strategy of user j which is the solution of BEST−REPLY$_j$ is given below.

BEST−REPLY$_j$ solution:

Assuming that computers are ordered in decreasing order of their available processing rates ($\mu_1^j \geq \mu_2^j \geq \cdots \geq \mu_n^j$), the solution $\mathbf{s}_j$ of the optimization problem BEST−REPLY$_j$ is given by

$$s_{ji} = \begin{cases} \frac{1}{\phi_j}\left(\mu_i^j - \sqrt{\mu_i^j}\,\frac{\sum_{i=1}^{c_j-1}\mu_i^j - \phi_j}{\sum_{i=1}^{c_j-1}\sqrt{\mu_i^j}}\right) & \text{if } 1 \leq i < c_j \\ 0 & \text{if } c_j \leq i \leq n \end{cases}$$

where c_j is the minimum index that satisfies the inequality

$$\sqrt{\mu_{c_j}^j} \leq \frac{\sum_{i=1}^{c_j}\mu_i^j - \phi_j}{\sum_{i=1}^{c_j}\sqrt{\mu_i^j}}$$

Let us see why.

At the Nash equilibrium the stability condition

$$\sum_{k=1}^{m} s_{ki}\,\phi_k < \mu_i,\ \ i = 1, \ldots, n,$$

is always satisfied because of

$$\mathbf{s}_j \in \underset{\tilde{\mathbf{s}}_j}{\operatorname{argmin}}\, D_j(\mathbf{s}_1, \ldots, \tilde{\mathbf{s}}_j, \ldots, \mathbf{s}_m)$$

and the fact that the total arrival rate does not exceed the total processing rate of the distributed system

$$\Phi \leq \Sigma_{i=1}^{n} \mu_i.$$

Thus the optimization problem $\mathrm{BEST{-}REPLY}_j$ may be considered with the only two restrictions,

$$s_{ji} \geq 0, \quad i = 1, \ldots, n,$$

$$\sum_{i=1}^{n} s_{ji} = 1.$$

Since $D_j(\mathbf{s}) = \sum_{i=1}^{n} s_{ji} F_i(\mathbf{s})$ with $F_i(\mathbf{s}) = \frac{1}{\mu_i - \Sigma_{k=1}^{m} s_{ki}\phi_k} = \frac{1}{\mu_i^j - s_{ji}\phi_j}$, we have

$$\frac{\partial D_j(\mathbf{s})}{\partial s_{ji}} = F_i(\mathbf{s}) + s_{ji}\phi_j F_i(\mathbf{s})^2 = \mu_i^j F_i(\mathbf{s})^2$$

and

$$\frac{\partial^2 D_j(\mathbf{s})}{(\partial s_{ji})^2} = 2\mu_i^j \phi_j F_i(\mathbf{s})^3 \geq 0$$

$$\frac{\partial^2 D_j(\mathbf{s})}{\partial s_{ji}\partial s_{jl}} = \frac{\partial}{\partial s_{jl}}\left(\frac{\partial D_j(\mathbf{s})}{\partial s_{ji}}\right) = 0, l \neq i.$$

Thus the Hessian of $D_j(\mathbf{s})$ is positive, which implies that $D_j(\mathbf{s})$ is a ***convex function*** of the load fractions of user j, $\mathbf{s}_j = (s_{j1}, s_{j2}, \ldots, s_{jn})$.

Since the constraints are all linear and they define a convex polyhedron, the ***set of feasible solutions*** of BEST−REPLY$_j$ is ***convex***.

Thus, the BEST−REPLY$_j$ problem involves minimizing a convex function over a convex feasible region and the first-order Kuhn–Tucker conditions are necessary and sufficient for optimality.

The Lagrangian is, with the Lagrange multipliers α and $\eta_i, i = 1, \ldots, n$,

$$L(s_{j1}, \ldots, s_{jn}, \alpha, \eta_1, \ldots, \eta_n) = \sum_{i=1}^{n} \frac{s_{ji}}{\mu_i^j - s_{ji}\phi_j} - \alpha\left(\sum_{i=1}^{n} s_{ji} - 1\right) - \sum_{i=1}^{n} \eta_i s_{ji}$$

The Kuhn–Tucker conditions imply that $s_{ji}, i = 1, \ldots, n$, is the optimal solution to the BEST−REPLY$_j$ problem if and only if there exist $\alpha \geq 0, \eta_i \geq 0, i = 1, \ldots, n$, such that

$$\frac{\partial L}{\partial s_{ji}} = 0,$$

$$\frac{\partial L}{\partial \alpha} = 0,$$

$$\eta_i s_{ji} = 0,\ \eta_i \geq 0,\ s_{ji} \geq 0,\ i = 1, \ldots, n.$$

Computing these conditions yields

$$\frac{\mu_i^j}{\left(\mu_i^j - s_{ji}\phi_j\right)^2} - \alpha - \eta_i = 0, \qquad i = 1, \ldots, n,$$

$$\sum_{i=1}^{n} s_{ji} = 1,$$

$$\eta_i s_{ji} = 0, \ \eta_i \geq 0, \ s_{ji} \geq 0, \ i = 1, \ldots, n.$$

These are equivalent to

$$\alpha = \frac{\mu_i^j}{\left(\mu_i^j - s_{ji}\phi_j\right)^2}, \qquad \text{if } s_{ji} > 0, \qquad 1 \leq i \leq n,$$

$$\alpha \leq \frac{\mu_i^j}{\left(\mu_i^j - s_{ji}\phi_j\right)^2}, \qquad \text{if } s_{ji} = 0, \qquad 1 \leq i \leq n,$$

$$\sum_{i=1}^{n} s_{ji} = 1, \quad s_{ji} \geq 0, \qquad i = 1, \ldots, n.$$

Clearly, a computer with a higher average processing rate should have a higher fraction of jobs assigned to it.

Thus, assuming (to simplify the presentation) that the indexing i of the computers is such that $\mu_1^j \geq \mu_2^j \geq \cdots \geq \mu_n^j$, the load fractions are ordered according to $s_{j1} \geq s_{j2} \geq \cdots \geq s_{jn}$.

It may therefore happen that the slow computers have no jobs assigned to them.

This means that there exists an index c_j $(1 \leq c_j \leq n)$ such that $s_{ji} = 0$ for $i = c_j, \ldots, n$.

Since $\dfrac{\mu_i^j}{\left(\mu_i^j - s_{ji}\phi_j\right)^2} = \alpha$ for $1 \leq i < c_j$, we have

$$\sqrt{\mu_i^j} = \sqrt{\alpha}\left(\mu_i^j - s_{ji}\phi_j\right) \text{ for } 1 \leq i < c_j,$$

which implies, after summation,

$$\sqrt{\alpha} = \frac{\sum_{i=1}^{c_j-1}\sqrt{\mu_i^j}}{\sum_{i=1}^{c_j-1}\mu_i^j - \sum_{i=1}^{c_j-1}s_{ji}\phi_j} = \frac{\sum_{i=1}^{c_j-1}\sqrt{\mu_i^j}}{\sum_{i=1}^{c_j-1}\mu_i^j - \phi_j}.$$

Since $\alpha \leq \frac{\mu_i^j}{\left(\mu_i^j - s_{ji}\phi_j\right)^2} = \frac{1}{\mu_i^j}$ for $i \geq c_j$, we have

$$\frac{\sum_{i=1}^{c_j-1}\sqrt{\mu_i^j}}{\sum_{i=1}^{c_j-1}\mu_i^j - \phi_j} \leq \frac{1}{\sqrt{\mu_{c_j}^j}}$$

hence $$\sqrt{\mu_{c_j}^j}\sum_{i=1}^{c_j-1}\sqrt{\mu_i^j} \leq \sum_{i=1}^{c_j-1}\mu_i^j - \phi_j$$

or, equivalently, $$\sqrt{\mu_{c_j}^j}\sum_{i=1}^{c_j}\sqrt{\mu_i^j} \leq \sum_{i=1}^{c_j}\mu_i^j - \phi_j \ .$$

Thus, assuming that the computers are ordered in decreasing order of their available processing rate, the index c_j so that $s_{ji} = 0$ for $i = c_j, \ldots, n$ is the minimum index that satisfies the above equation.

Moreover, for $1 \leq i < c_j$, $s_{ji} = \frac{1}{\phi_j}\left(\mu_i^j - \frac{\sqrt{\mu_i^j}}{\sqrt{\alpha}}\right) = \frac{1}{\phi_j}\left(\mu_i^j - \sqrt{\mu_i^j}\frac{\sum_{i=1}^{c_j-1}\mu_i^j - \phi_j}{\sum_{i=1}^{c_j-1}\sqrt{\mu_i^j}}\right)$.

It can be easily checked that the following algorithm determines user j's best reply strategy.

BEST−REPLY$(\mu_1^j, \mu_2^j, \ldots, \mu_n^j, \phi_j)$

Input: Available processing rates: $\mu_1^j, \mu_2^j, \ldots, \mu_n^j$;
Total arrival rate: ϕ_j

Output: Load fractions: $s_{j1}, s_{j2}, \ldots, s_{jn}$;

1. Sort the computers in decreasing order of their available processing rates $(\mu_1^j \geq \mu_2^j \geq \cdots \geq \mu_n^j)$
2. $t \leftarrow \dfrac{\sum_{i=1}^{n} \mu_i^j - \phi_j}{\sum_{i=1}^{n} \sqrt{\mu_i^j}}$
3. **while** $(t \geq \sqrt{\mu_i^n})$ **do**

$$s_{jn} \leftarrow 0 \; ; n \leftarrow n-1 \; ; t \leftarrow \frac{\sum_{i=1}^{n} \mu_i^j - \phi_j}{\sum_{i=1}^{n} \sqrt{\mu_i^j}}$$

4. **for** $i = 1, \ldots, n$ **do**

$$s_{ji} \leftarrow \frac{(\mu_i^j - t\sqrt{\mu_i^j})}{\phi_j}$$

Because of the sorting procedure in Step 1, the algorithm's execution time is $O(n \log n)$.

Application example:
A system has 3 computers and only one user.

The computers have the available processing rates: $\mu_1^1 = 10.0$, $\mu_2^1 = 2.0$ and $\mu_3^1 = 1.0$.
The total arrival rate is $\phi_1 = 6.0$.

- The computers are already sorted in decreasing order of their processing rate.
- Execution of Step 2 gives

$$t = \frac{10 + 2 + 1 - 6}{\sqrt{10} + \sqrt{2} + \sqrt{1}} = 1.255\ .$$

- The while loop in Step 3 is executed because $t > \sqrt{\mu_3^1}$. In this loop $s_{13} = 0, n = 2$ and t is updated to $1.311 < \sqrt{\mu_2^1}$.
- Then the algorithm proceeds to Step 4. In this step the values of the load fractions are computed: $s_{11} = 0.975$ and $s_{12} = 0.025$.

A distributed load balancing algorithm

The computation of the Nash equilibrium requires some coordination between the users, in particular in order to obtain the load information from each computer.

This can be achieved by using distributed greedy best reply algorithms, where each user updates from time to time its load balancing strategy by computing the best reply against the existing load balancing strategies of the other users.

A greedy best reply algorithm for computing the Nash equilibrium, based on the BEST−REPLY algorithm and with users synchronized such that they update their strategies in a round-robin fashion, is presented next.

The execution of this algorithm is restarted periodically or when the system parameters are changed. Once the Nash equilibrium is reached, the users will continue to use the same strategies and the system remains in equilibrium. This equilibrium is maintained until a new execution of the algorithm is initiated.

j—the user number;
l—the iteration number;
$\mathbf{s}_j^{(l)}$—the strategy of user j computed at iteration l;
$D_j^{(l)}$—user j's expected execution time at iteration l;
ε—a properly chosen acceptance tolerance;

$norm$—the L^1-norm at iteration l, defined as $norm = \sum_{j=1}^{m} |D_j^{(l-1)} - D_j^{(l)}|$
Send$(j, (p, q, r))$—send the message (p, q, r) to user j;
Recv$(j, (p, q, r))$—receive the message (p, q, r) from user j;
(where p is a real number, and q, r are integer numbers).

User j, $(j = 1, \ldots, m)$ executes :
1. Initialization :
$\mathbf{s}_j^{(0)} \leftarrow \mathbf{0}$;
$\mathbf{D}_j^{(0)} \leftarrow \mathbf{0}$;
$l \leftarrow 0$;
$norm \leftarrow 1$;
$sum \leftarrow 0$;
$tag \leftarrow$ CONTINUE;
$left = [(j - 2) \bmod m] + 1$;
$right = [j \bmod m] + 1$;
2. while (1) **do**
if$(j = 1)$ {user 1}
if $(l \neq 0)$
Recv$(left, (norm, l, tag))$;
if $(norm < \varepsilon)$
Send$(right, (norm, l,$ STOP$))$;
exit;
$sum \leftarrow 0$;
$l \leftarrow l + 1$;
else {the other users}
Recv$(left, (sum, l, tag))$;
if $(tag =$ STOP$)$
if $(j \neq m)$ **Send**$(right, (sum,$ $l,$ STOP$))$;
exit;
for $i = 1, \ldots, n$ **do**
Obtain μ_i^j by inspecting the run queue of each computer
$(\mu_i^j \leftarrow \mu_i - \sum_{k=1, k \neq j}^{m} s_{ki} \phi_k)$;
$\mathbf{s}_j^{(l)} \leftarrow$ BEST $-$ REPLY $(\mu_1^j, \ldots, \mu_n^j, \phi_j)$;
Compute $D_j^{(l)}$;
$sum \leftarrow sum + |D_j^{(l-1)} - D_j^{(l)}|$;
Send$(right, (sum, l,$ CONTINUE$))$;
endwhile

In multiprogrammed heterogeneous distributed systems, the NASH scheme works as follows.

Each user has an associated ***scheduler agent*** (process) which makes the allocation decisions and communicates with the scheduling agents of the other users in the system.
The NASH algorithm is executed periodically by this ***set of scheduling agents***.

The scheduling agent estimates the job arrival rate ϕ_j at the user by considering the number of arrivals over a fixed interval of time.
It also queries the state of each computer in the system and based on the estimated available processing rate $\mu_i - \Sigma_{k=1}^{m} s_{ki}\phi_k$ reported by the computers it decides the fractions s_{ji}.
Once the fractions are determined the scheduling agent sends the next job to computer i with probability s_{ji}.

An important practical question is whether such ‘best reply’ algorithms converge to the Nash equilibrium.

Results about the convergence of such algorithms have been obtained in the context of routing in parallel links.
These studies have been limited to special cases of two parallel links shared by two users or by $m \geq 2$ users but with linear cost links.

For M/M/1-type cost functions there is no known proof that such algorithms converge for more than two users.
As shown by several experiments done on different settings, these algorithms may converge for more than two users. Such experiments that confirm this hypothesis are presented next.

The convergence proof for more than two users is still an open problem.

Convergence of the NASH algorithm

An important issue related to the greedy best reply algorithm is the dynamics of reaching the equilibrium.

The variant NASH_0 of NASH algorithm uses $\mathbf{s}(0) = \mathbf{0}$ as the initialization step. This initialization step is an obvious choice but it may not lead to a fast convergence to the equilibrium.

The variant NASH_P of the algorithm consists in replacing the initialization step by

1. **Initialization:**

   for $i = 1, \ldots, n$ **do**

   $s_{ji}^{(0)} \leftarrow \frac{\mu_i}{\sum_{k=1}^{n} \mu_k}$;

   $\mathbf{D}_j^{(0)} \leftarrow \mathbf{0}$;

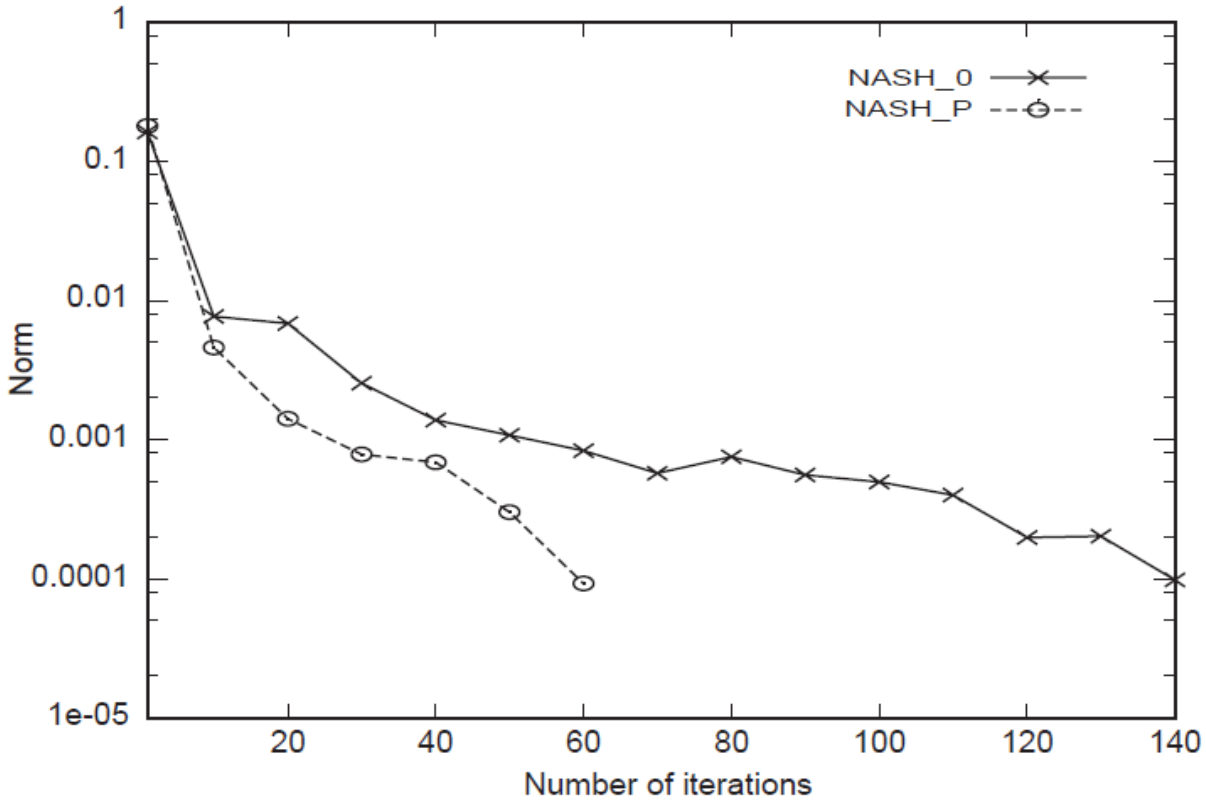
   $l \leftarrow 0$;

   $\vdots$

Using this initialization the starting point will be a proportional allocation of jobs to computers according to their processing rate.

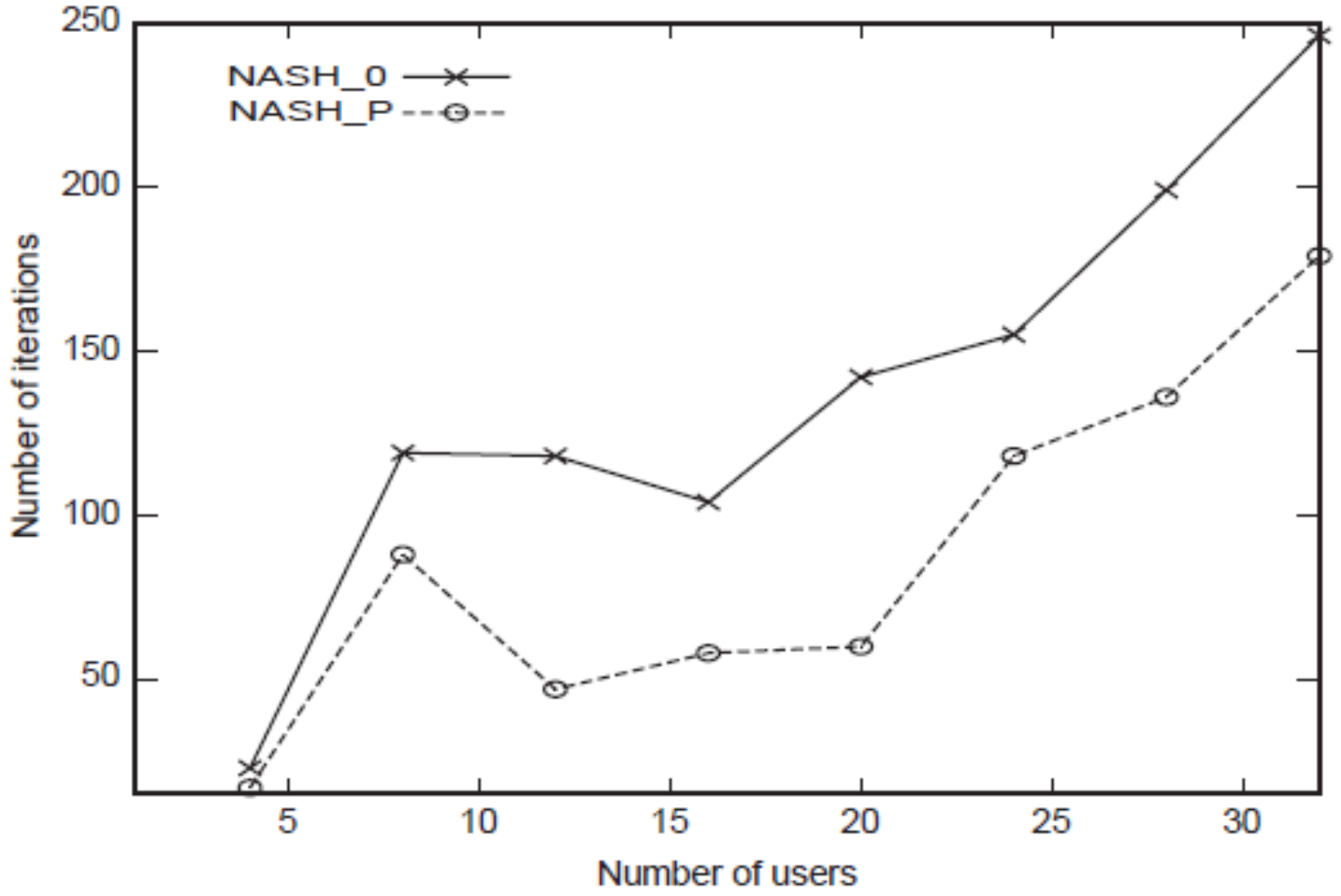
We expect a better convergence using NASH_P instead of NASH_0.

The norm vs. the number of iterations is shown for a system with 16 computers shared by 10 users.



The NASH_P algorithm significantly outperforms NASH_0 algorithm.
The intuitive explanation for this performance is that the initial proportional allocation is close to the equilibrium point and the number of iterations needed to reach the equilibrium is reduced.
Using the NASH_P algorithm the number of iterations needed to reach the equilibrium is reduced by more than a half compared with NASH_0.

To study the influence of the number of users on the convergence of both algorithms, the number of iterations needed to reach the equilibrium ($norm < 10^{-4}$) is plotted for a system with 16 computers and a variable number of users (from 4 to 32).



NASH_P significantly outperforms NASH_0, reducing the number of iterations needed to reach the equilibrium in all the cases.

Experimental results

The simulation model consists of a collection of computers connected by a communication network. Jobs arriving at the system are distributed to the computers according to the specified load balancing scheme.

Jobs which have been dispatched to a particular computer are *run-to-completion* (i.e. no preemption) in first-come-first-served (FCFS) order.

Each computer is modeled as an M/M/1 queueing system.

The main performance metrics used are the *expected response time* and the *fairness index*.

The *fairness index* is

$$I(\mathbf{D}) = \frac{(\sum_{j=1}^{m} D_j)^2}{m \sum_{j=1}^{m} {D_j}^2}$$

where the parameter $\mathbf{D}$ is the vector $\mathbf{D} = (D_1, D_2, \ldots, D_m)$ where D_j is the expected execution time of user *j*'s jobs. This index is a measure of the 'equality' of users' job execution times. If all the users have the same expected job execution times then $I = 1$ and the system is 100% fair to all users and it is load balanced. If the differences on D_j increase, I decreases and the load balancing scheme favors only some users.

Three static load balancing schemes are compared with the NASH algorithm:

• *Global optimal scheme* (*GOS*).

This scheme minimizes the expected execution time over all jobs executed by the system. The load fractions (**s**) are obtained by solving the following nonlinear optimization problem:

$$\min_{\mathbf{s}} \frac{1}{\Phi} \sum_{k=1}^{m} \phi_j D_j(\mathbf{s})$$

subject to the constraints

$$s_{ji} \geq 0, \qquad i = 1, \ldots, n,$$

$$\sum_{i=1}^{n} s_{ji} = 1, \qquad j = 1, \ldots, m,$$

$$\sum_{k=1}^{m} s_{ki}\phi_k < \mu_i, \qquad i = 1, \ldots, n.$$

This scheme provides the overall optimum for the expected execution time but it is not user-optimal and is unfair. This is a centralized scheme.

• *Individual optimal scheme* (*IOS*).

In this scheme, each job optimizes its response time for itself independently of others. In general, the solution given by this scheme is not optimal and in some cases we expect worse response time than the other policies. It is based on an iterative procedure that is not very efficient. The advantage of this scheme is that it provides a fair allocation. This is a centralized scheme.

• *Proportional scheme* (*PS*).

According to this scheme each user allocates its jobs to computers in proportion to their processing rate. This allocation seems to be a natural choice but it may not minimize the user's expected response time or the overall expected response time. The fairness index for this scheme is always 1 as can be easily seen from the index definition. Like NASH, this is a distributed scheme.

To study the effect of system utilization, a heterogeneous system consisting of 16 computers with four different processing rates is simulated. This system is shared by 10 users.

System configuration:

Relative processing rate	1	2	5	10
Number of computers	6	5	3	2
Processing rate (jobs/s)	10	20	50	100

Job arrival fractions ϕ_j/Φ for each user:

User	1	2	3-6	7	8-10
ϕ_j/Φ	0.3	0.2	0.1	0.07	0.01

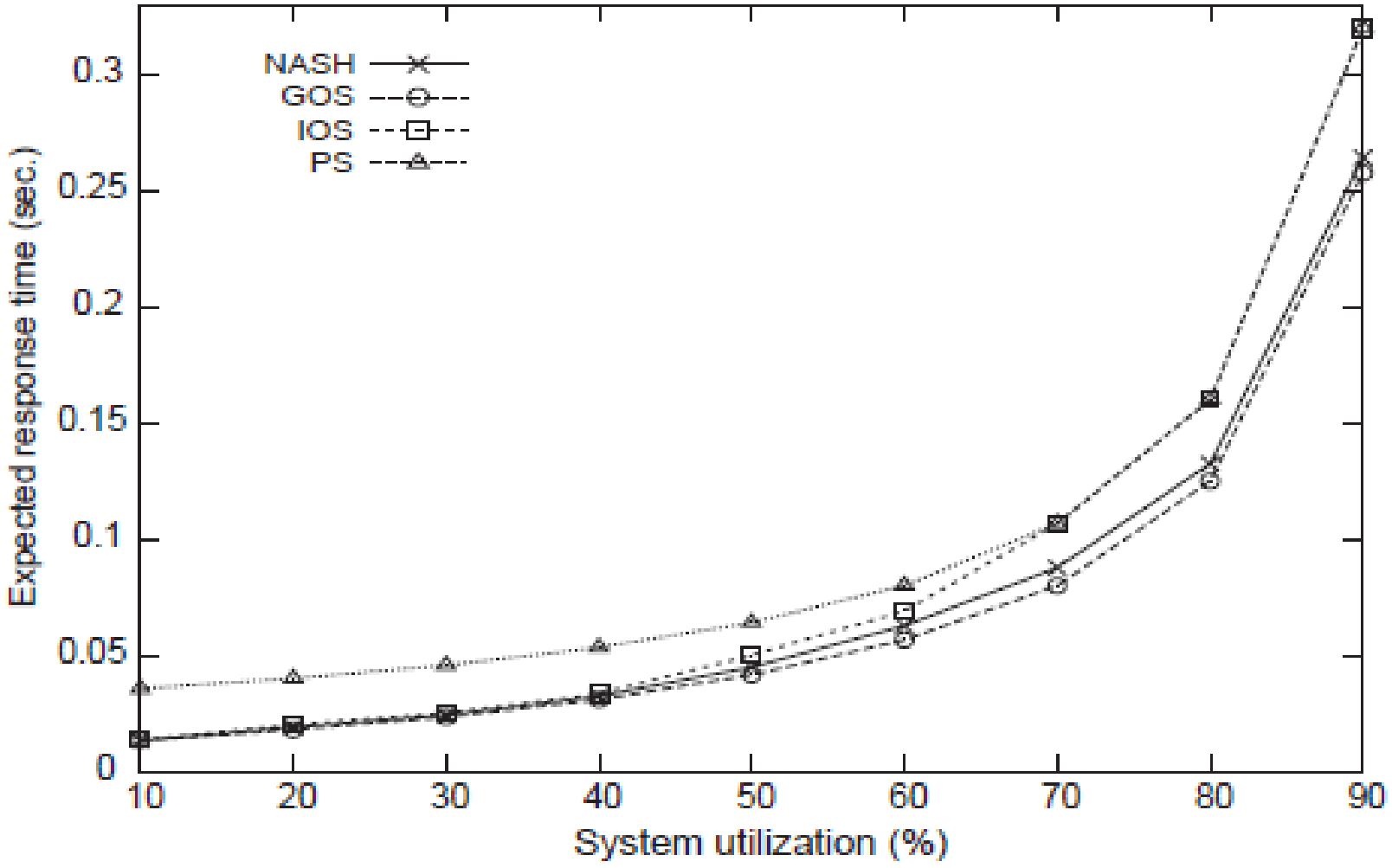
Only computers that are at most 10 times faster than the slowest are considered because this is the case in most of the current heterogeneous distributed systems.

For each experiment the total job arrival rate in the system is determined by the system utilization and the aggregate processing rate of the system. *System utilization* (ρ) is defined as the ratio of the total arrival rate to the aggregate processing rate of the system

$$\rho = \frac{\Phi}{\sum_{i=1}^{n} \mu_i}$$

Choosing a fixed value for the system utilization, the total job arrival is easily determined. For example, for $\rho = 10$ % and an aggregate processing rate of 510 jobs/s then the arrival rate in the system is $\lambda = 51$ jobs/s. Multiplying the arrival rate by the job fractions gives the job arrival rates ϕ_j for the users.

Expected response time of the system for different values of system utilization (ranging from 10% to 90%).

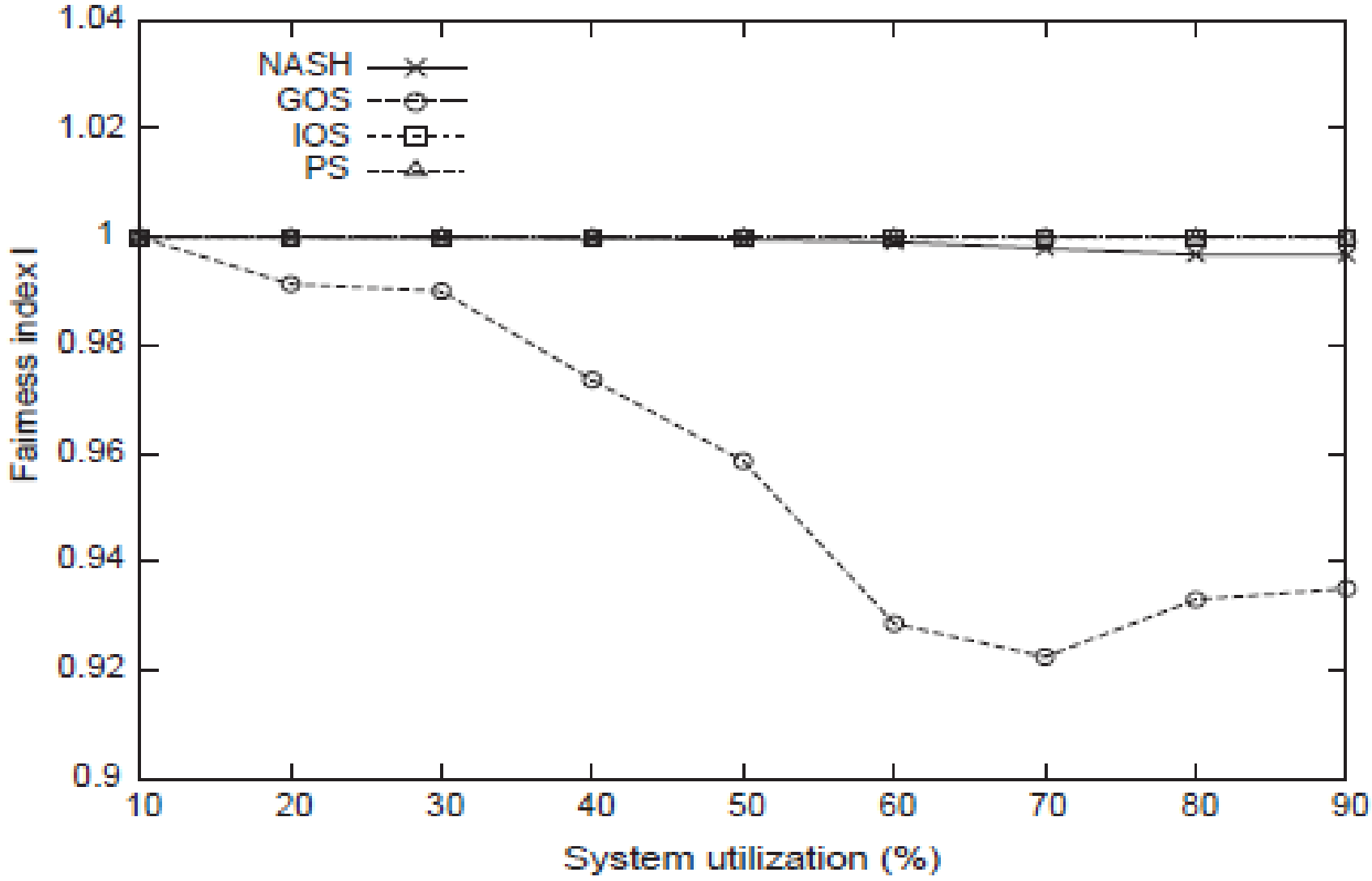


At low loads (from 10% to 40%) all the schemes except PS yield almost the same performance. The poor performance of the PS scheme is due to the fact that the less powerful computers are significantly overloaded.

At medium loads (from 40% to 60%) NASH scheme performs significantly better than PS and approaches the performance of GOS.
For example at load level of 50% the mean response time of NASH is 30% less than PS and 7% greater than GOS (this is the price of anarchy).

At high loads IOS and PS yield the same expected response time which is greater than that of GOS and NASH.
The expected response time of NASH scheme is very close to that of GOS.

Fairness index for different values of system utilization (ranging from 10% to 90%).



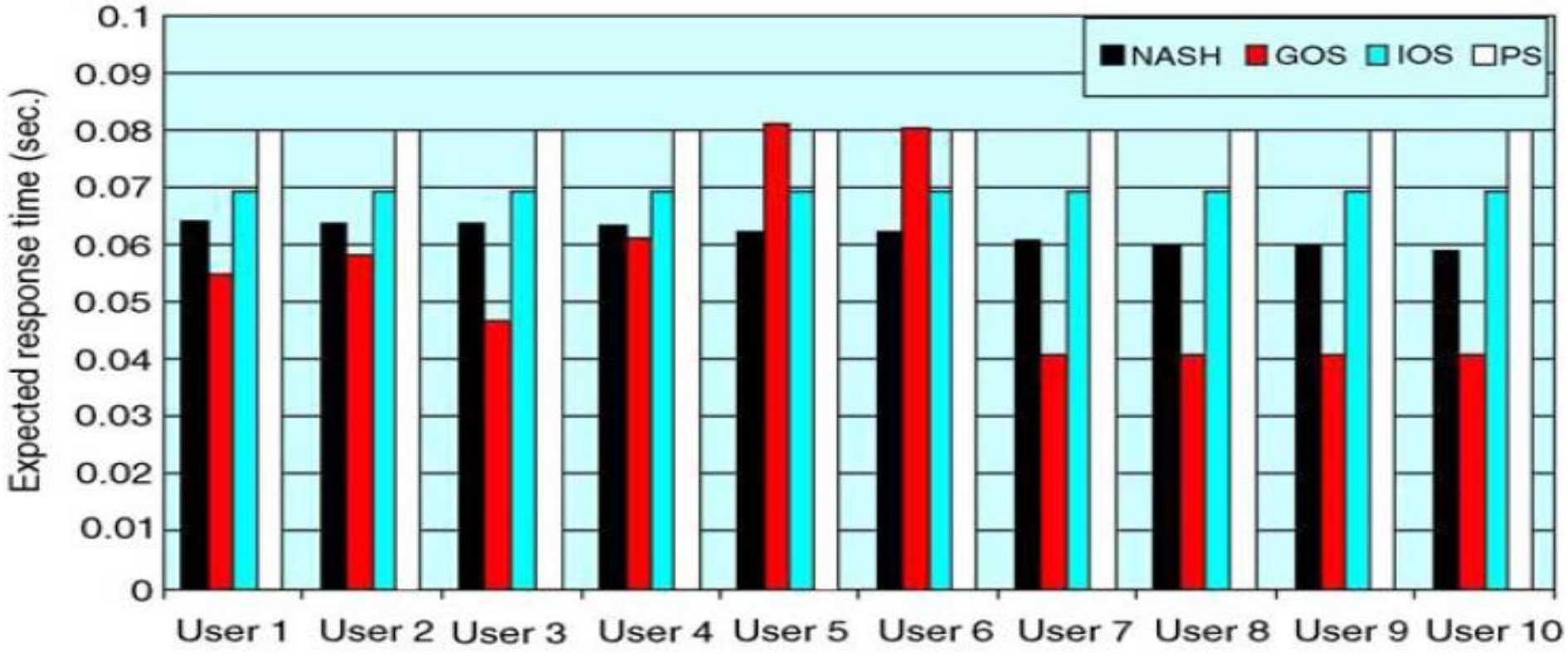
The fairness index of GOS varies from 1 at low load, to 0.92 at high load.

The IOS and PS schemes maintain a fairness index of 1 over the whole range of system loads. Recall that the PS scheme has a fairness index of 1 independent of the system load.

The NASH scheme has a fairness index close to 1 and each user obtains the minimum possible expected response time for its own jobs given what every other user is doing (i.e. it is user-optimal).

The stability of the allocation under noncooperative behavior and decentralization are the main advantages of NASH scheme.

An interesting issue is the impact of static load balancing schemes on individual users. The figure presents the expected response time for each user considering all static schemes at medium load (=60%).



Expected response time for each user.

NASH scheme provides the minimum possible expected execution time for each user (according to the properties of the Nash equilibrium).

In the case of GOS scheme there are large differences in users' expected execution times.

The IOS and PS schemes guarantee equal expected response times for all users but with the disadvantage of a higher expected execution time for their jobs.