Experimental Results for Stackelberg Scheduling Strategies*

A.C. Kaporis[1], L.M. Kirousis[1,2], E.I. Politopoulou[1,2], and P.G. Spirakis[1,2]

[1] Department of Computer Engineering and Informatics,
University of Patras, Greece
{kaporis, kirousis, politop, spirakis}@ceid.upatras.gr
[2] Research Academic Computer Technology Institute,
P.O Box 1122, 26110 Patras, Greece
{politop, spirakis}@cti.gr

Abstract. In large scale networks users often behave selfishly trying to minimize their routing cost. Modelling this as a noncooperative game, may yield a *Nash* equilibrium with unboundedly poor network performance. To measure this inefficacy, the *Coordination Ratio* or *Price of Anarchy* (*PoA*) was introduced. It equals the ratio of the cost induced by the worst Nash equilibrium, to the corresponding one induced by the overall optimum assignment of the jobs to the network. On improving the *PoA* of a given network, a series of papers model this selfish behavior as a *Stackelberg* or *Leader-Followers* game.

We consider random tuples of machines, with either linear or M/M/1 latency functions, and *PoA* at least a *tuning parameter* c. We validate a variant (`NLS`) of the *Largest Latency First* (`LLF`) Leader's strategy on tuples with $PoA \geq c$. `NLS` experimentally improves on `LLF` for systems with inherently high *PoA*, where the Leader is *constrained* to control low portion α of jobs. This suggests even better performance for systems with arbitrary *PoA*. Also, we bounded experimentally the least Leader's portion α_0 needed to induce optimum cost. Unexpectedly, as parameter c increases the corresponding α_0 decreases, for M/M/1 latency functions. All these are implemented in an extensive *Matlab* toolbox.

1 Introduction

We consider the problem of system resource allocation [28]. This problem is one of the basic problems in system management, though systems today have high availability of bandwidth and computational power.

* The 2nd and 4th author are partially supported by Future and Emerging Technologies programme of the EU under EU contract 001907 "*Dynamically Evolving, Large Scale Information Systems (DELIS)*". The 1st, 2nd and 3rd author are partially supported by European Social Fund (ESF), Operational Program for Educational and Vacational Training II (EPEAEK II), and particularly *PYTHAGORAS*.

S.E. Nikoletseas (Ed.): WEA 2005, LNCS 3503, pp. 77–88, 2005.
© Springer-Verlag Berlin Heidelberg 2005

Such systems are large scale networks, for example broadband [27], wireless [8] and peer to peer networks[7] or Internet. The users have the ability to select their own route to their destination with little or no limitation [4, 20, 9]. Since the users are instinctively selfish, they may use the right of path selection and may select a route that maximizes their profit. This selfish routing behavior can be characterized by a fixed state, which in game theory is called *Nash Equilibrium* [15, 18]. In this context, the interested reader can find much of theoretic work in [15, 12, 14, 9, 11, 19, 20, 21].

However, Nash Equilibrium may lead a system to suboptimal behavior. As a measure of how worse is the Nash equilibrium compared to the overall system's optimum, the notion of *coordination ratio* was introduced in [12, 14]. Their work have been extended and improved (*price of anarchy* here is another equivalent notion) in [14, 24, 3, 22, 23, 6, 4].

Selfish behavior can be modeled by a *non-cooperative game*. Such a game could impose strategies that might induce an equilibrium closer to the overall optimum. These strategies are formulated through pricing mechanisms[5], algorithmic mechanisms[16, 17] and network design[25, 10]. The network administrator or designer can define prices, rules or even construct the network, in such a way that induces near optimal performance when the users selfishly use the system.

Particulary interesting is the approach where the network manager takes part to the non-cooperative game. The manager has the ability to control centrally a part of the system resources, while the rest resources are used by the selfish users. This approach has been studied through *Stackelberg* or *Leader-Follower* games [2, 23, 9, 11, 26]. The advantage of this approach is that it might be easier to be deployed in large scale networks. This can be so, since there is no need to add extra components to the network or, to exchange information between the users of the network.

Let us concentrate on the setting of this approach. The simplified system consists of a set of machines with load depended latency functions and a flow of jobs with rate r. The manager controls a fraction α of the flow, and assigns it to machines in a way that the induced cost by the users is near or equals the overall optimal. An interesting issue investigated in [23, 9], is how should the manager assign the flow he controls into the system, as to induce optimal cost by the selfish users. For the case of linear load functions, in [23] was presented a polynomial algorithm (`LLF`) of computing a strategy with cost at most $\frac{4}{3+\alpha}$ times the overall optimum one, where α is the fraction of the rate that the manager controls. Korilis et al [9] has initiated this game theoretic approach and investigated on the necessary conditions such that the manager's assignment induces the optimum performance on a system with M/M/1 latency functions.

1.1 Motivation and Contribution

Our work is motivated by the work in [1, 23, 9]. We consider a simple modification of the algorithm *Largest Latency First* (`LLF`) [23] called *New Leader Strategy*

(NLS). Experiments suggest that NLS has better performance in competitive systems of machines, that is, systems with high value of price of anarchy PoA. Also, it has good performance in cases where the Leader may be constrained to use a small portion α of flow. Notice that $PoA \leq 4/3$ for linear latency functions. Furthermore, a highly nontrivial algorithm presented in [1] slightly improves over LLF for the case of linear latency functions. Then, despite its simplicity, our heuristic has comparatively good performance.

Additionally, we conducted thousands random tuples of machines, with either linear or M/M/1 latency functions. We experimentally tried to compute an upper bound α_0 for the least possible portion of flow that a Leader needs to induce overall optimal behavior. We have considered tuples of machines with M/M/1 latency functions such that their price of anarchy is at least a parameter c. Surprisingly, as parameter c increases (resulting to more competitive systems of machines), the average value of α_0 *decreases*.

2 Improved Stackelberg Strategies

2.1 Model - Stackelberg Strategies

For this study the model and notation of [23] is used. We consider a set M of m machines, each with a latency function $\ell(\cdot) \geq 0$ continuous and nondecreasing, that measures the load depended time that is required to complete a job. Jobs are assigned to M in a finite and positive rate r. Let the m-vector $X \in \mathcal{R}_+^m$ denote the assignment of jobs to the machines in M such that $\sum_{i=1}^m x_i = r$. The latency of machine i with load x_i is $\ell_i(x_i)$ and incurs cost $x_i\ell_i(x_i)$, convex on x_i. This instance is annotated (M, r). The *Cost* of an assignment $X \in \mathcal{R}_+^m$ on the (M, r) instance is $C(X) = \sum_{i=1}^m x_i\ell_i(x_i)$, measuring system's performance. The minimum cost is incurred by a unique assignment $O \in \mathcal{R}_+^m$, called the *Optimum* assignment. The assignment $N \in \mathcal{R}_+^m$ defines a *Nash equilibrium*, if no user can find a loaded machine with lower latency than any other loaded machine. That is, all machines i with load $n_i > 0$ have the same latency L while any machine j with load $n_j = 0$ has latency $L_j \geq L$. According to the Definition 2.1 in [23]:

Definition 1. *An assignment $N \in \mathcal{R}_+^m$ to machines M is at Nash equilibrium (or is a Nash assignment) if whenever $i, j \in M$ with $n_i > 0$, $\ell_i(n_i) \leq \ell_j(n_j)$.*

The Nash assignment N causes cost $C(N)$ commonly referred to as *Social Cost* [15, 12, 14, 9, 11, 19, 20, 21]. The social cost $C(N)$ is higher than the optimal one $C(O)$, leading to a degradation in system performance. The last is quantified via the *Coordination Ratio*[12, 14, 3] or *Price of Anarchy (PoA)* [24], i.e. the worst-possible ratio between the social cost and optimal cost: $PoA = \frac{C(N)}{C(O)}$, and the goal is to minimize PoA [1] To do so, a *hierarchical non cooperative Leader-Follower* or *Stackelberg game* is used [2, 23, 9, 11, 26]. In such a game, there is a

[1] Notice that in a general setting may exist a set A of Nash equilibria, then PoA is defined with respect to worst one, i.e. $PoA = \max_{N \in A} \frac{C(N)}{C(O)}$.

set M of machines, jobs with flow rate r and a distinguished player or *Leader* who is responsible for assigning centrally an α portion of the rate r to the system so as to decrease the total social cost of the system. The rest of the players, called *Followers* are assigning selfishly the remaining $(1-\alpha)r$ flow in order to minimize their individual cost. This instance is called *Stackelberg instance* and is annotated by (M, r, α). The Leader assigns $S \in \mathcal{R}_+^m$ to M and the Followers react, inducing an assignment in Nash equilibrium. We give the same definition for an *induced assignment at Nash Equilibrium* or *induced Nash assignment* as in Definition 2.7 of [23].

Definition 2. *Let $S \in \mathcal{R}_+^m$ be a Leader assignment for a Stackelberg instance (M, r, α) where machine $i \in M$ has latency function ℓ_i, and let $\widetilde{\ell}_i(x_i) = \ell_i(s_i + x_i)$ for each $i \in M$. An equilibrium induced by S is an assignment $T \in \mathcal{R}_+^m$ at Nash equilibrium for the instance $(M, (1-a)r)$ with respect to latency function $\widetilde{\ell}$. We then say that $S + T$ is an assignment induced by S for (M, r, α).*

The goal is achieved if $C(S+T) \simeq C(O)$.

We consider here two types of latency functions: linear and M/M/1. Linear latency functions have the form $\ell_i(x_i) = a_i x_i + b_i,\ i \in M,\ X \in \mathcal{R}_+^m$ and it holds $PoA \leq \frac{4}{3}$. M/M/1 latency functions have the form $\ell_i(x_i) = \frac{1}{u_i - x_i},\ i \in M,\ X \in \mathcal{R}_+^m$ and it holds $PoA \leq \frac{1}{2}\left(1 + \sqrt{\frac{u_{min}}{u_{min} - R_{max}}}\right)$, where u_{min} is the smallest allowable machine capacity and R_{max} is the largest allowable traffic rate.

Finally, to tune the competitiveness of a particular system M, we define the parameter c as a lower bound of its PoA. Thus, systems with highly valued parameter c are particularly competitive.

2.2 Algorithm NLS

Algorithm: NLS(M, r, α)
Input: Machines $M = \{M_1, \ldots, M_m\}$, flow r, and portion $\alpha \in [0, 1]$
Output: An assignment of the load αr to the machines in M.
begin:
  Compute the global Optimum assignment $O = \langle o_1, \ldots, o_m \rangle$ of flow r on M;
  Compute the Nash assignment $N = \langle n_1, \ldots, n_m \rangle$ of the flow $(1-\alpha)r$ on M;
  Let $M^* = \{M_i \in M \mid n_i = 0\}$;
  If $\sum_{\{i: M_i \in M^*\}} o_i \geq \alpha r$ **then** assign local optimally the flow αr on M^*;
   else assign the flow αr on M according to LLF;
  end if;
end;

Notice that it is possible to heuristically compute an even larger subset M^* unaffected by the Followers, allowing us to assign to it a even larger portion $\alpha' > \alpha$ of flow.

In [23] it was presented the *Large Latency First* (LLF) Stackelberg strategy for a *Leader* that controls a portion α of the total flow r of jobs, to be scheduled to

a system of machines M. For the case of machines with linear latency functions, it was demonstrated that the induced scheduling cost is at most $\frac{4}{3+\alpha}$ of the optimum one.

We present and validate experimentally the *New Leader Strategy* (`NLS`). Our motivation was a system of machines presented in [23], end of page 17. In that example, the set of machines is $M = \{M_1, M_2, M_3\}$ with corresponding latency functions $\ell_1(x) = x, \ell_2(x) = x + 1, \ell_3(x) = x + 1$. `LLF` at first computes the optimum assignment $O = \langle o_1 = \frac{4}{6}, o_2 = \frac{1}{6}, o_3 = \frac{1}{6}\rangle$, of the total flow $r = 1$ to the given set of machines M. On each machine i, load o_i incurs latency value $\ell_i(o_i),\ i = 1, \ldots, 3$. Then, `LLF` indexes the machines, from lower to higher latency values, computed at the corresponding optimum load. In this example, the initial indexing remains unchanged, since: $\ell_1(o_1) \leq \ell_2(o_2) \leq \ell_3(o_3)$. In the sequel, it computes a Stackelberg scheduling strategy $S = \langle s_1, s_2, s_3\rangle$ for the Leader as follows. `LLF` schedules the flow αr that Leader controls, filling each machine i up to its optimum load o_i, proceeding in a "largest latency first" fashion. At first, machine 3 is assigned a flow at most its optimum load o_3. If $\alpha r - o_3 > 0$, then machine 2 is assigned a flow at most its optimum load o_2. Finally, if $\alpha r - o_3 - o_2 > 0$, then machine 1 receives at most its optimum load o_1. Notice that all selfish followers prefer the first machine, i.e the Nash assignment is $N = \langle n_1 = 1, n_2 = 0, n_3 = 0\rangle$, since the total flow equals $r = 1$. Provided that no Follower affects the load assignment S of the Leader to the subset of machines $M' = \{2, 3\}$, a crucial observation is that strategy S computed by `LLF` is not always optimal. It is optimal only in the case that the portion α of Leader equals: $\alpha = \frac{o_2+o_3}{r}$. In other words, the assignment of the Leader would be optimal if its flow was enough to fill all machines 2 and 3 up to their corresponding optimal loads o_2, o_3. Taking advantage of this, a better Stackelberg strategy is: $S' = \langle s'_1 = 0,\ s'_2 = o^*_2,\ s'_3 = o^*_3\rangle$, where o^*_2 and o^*_3 are the corresponding *local* optimum loads, of the flow αr that a Leader controls, on the subset of the machines $\{2, 3\}$ which are not appealing for the Followers.

To illustrate this, consider any $\alpha < o_2 + o_3 = \frac{1}{6} + \frac{1}{6}$, for example let $\alpha = \frac{1}{6}$. Then `LLF` computes the Stackelberg strategy $S = \langle 0, 0, \frac{1}{6}\rangle$, inducing the Nash assignment $N = \langle \frac{5}{6}, 0, \frac{1}{6}\rangle$ with cost $C_S = \left(\frac{5}{6}\right)^2 + \left(1 + \frac{1}{6}\right)\frac{1}{6} = \frac{8}{9}$. However, the *local* optimum assignment of the flow $\alpha = \frac{1}{6}$ to machines 2 and 3 is $S' = \langle 0, \frac{1}{12}, \frac{1}{12}\rangle$. This induces the Nash assignment $N' = \langle \frac{5}{6}, \frac{1}{12}, \frac{1}{12}\rangle$ with cost $C_{S'} = \left(\frac{5}{6}\right)^2 + \left(1 + \frac{1}{12}\right)\frac{1}{6} = \frac{7}{8} < \frac{8}{9}$.

We propose algorithm `NLS` that takes advantage all these issues discussed above. Intuitively, it tries to compute a *maximal* subset $M^* \subseteq M = \{1, \ldots, m\}$ of machines not appealing to the selfish users. This subset $M^* \subseteq M$ consists of exactly those machines that receive no flow by the Nash assignment of $(1 - \alpha)r$ flow on M. Then it assigns the portion αr of a Leader local optimally on M^*. The empirical performance of `NLS` is presented in Section 4, in Figures 2 and 4.

3 Algorithm OpTop

3.1 Description

In [23] (also see the important results in [9] for the case of M/M/1 latency functions) it was possed the important question:

> "Compute the minimum flow of jobs that a Leader may play according to a Stackelberg scheduling strategy to a system of machines, in a way that the selfish play of the Followers leads the system to its optimum cost."

In this section, we investigate this issue experimentally for the case of machines with linear latency functions. Algorithm OpTop below (based on features of LLF), tries to control a minimal portion α of the total flow r of jobs. It schedules this flow to a system of m machines, in a way that the selfish play of the Followers drives the system to its optimum cost. Intuitively, OpTop tries to find a small subset of machines that have the following stabilizing properties:

- The selfish play of the Followers will not affect the flow αr assigned by the Leader optimally to these machines.
- The selfish play of the Followers of the remaining $(1-\alpha)r$ flow on the remaining machines will drive the system to its optimum cost.

Algorithm: OpTop $(M,\ r,\ r_0)$
Input: Machines $M = \{M_1, \ldots, M_m\}$, flow r, initial flow r_0
Output: A portion α of flow rate r_0
begin:
  Compute the Nash assignment $N := \langle n_1, \ldots, n_m \rangle$ of flow r on machines M;
  Compute the Optimum assignment $O := \langle o_1, \ldots, o_m \rangle$ of flow r on machines M;
  If $(N \equiv O)$ **return** $(r_0 - r)/r_0$;
    else $(M, r) \leftarrow$ Simplify(M, r, N, O);
    return OpTop(M, r, r_0);
  end if;
end;
Procedure: Simplify(M, r, N, O)
Input: Machines $M = \{M_1, \ldots, M_m\}$, flow r
    Nash assignment $N := \langle n_1, \ldots, n_m \rangle$
    Optimum assignment $O := \langle o_1, \ldots, o_m \rangle$
Output: Set of machines M, Flow r
begin:
  for $i = 1$ **to** size(M) **do:**
    If $o_i \geq n_i$ **then**
      $r \leftarrow r - o_i$;
      $M \leftarrow M \backslash \{M_i\}$;
    end if;
  end for;
end;

The key features of OpTop are presented with the help of Figures 1a, 1b, 1c. The corresponding Nash and Optimum assignments to these machines are de-

noted as: $N = \langle n_1, \ldots, n_5 \rangle$, such that $\sum_{i=1}^{5} n_i = r$, $O = \langle o_1, \ldots, o_5 \rangle$, such that $\sum_{i=1}^{5} o_i = r$.

Definition 3. *Machine i is called* over-loaded *(or* under-loaded*) if $n_i > o_i$ (or $n_i < o_i$). Machine i is called optimum-loaded if $n_i = o_i, i = 1, \ldots, m$.*

Initially, the algorithm assigns to all under-loaded machines in Figure 1a their optimum load. That is, it assigns optimum load o_4 and o_5 to machines 4 and 5 in Figure 1b. Then the players selfishly assign the remaining $r - o_4 - o_5$ flow to the system of 5 machines. Observe that in the induced Nash assignment, none of the machines 4 and 5 receives flow. That is, machines 4 and 5 have been stabilized to their optimum load, *irrespectively* of the selfish behavior of the Followers, see also Theorem 1.

A crucial point is that we can remove machines 4 and 5 from consideration and run recursively `OpTop` on the simplified system of machines. In other words, the induced game now is equivalent to scheduling the remaining $r - o_4 - o_5$ flow to the remaining machines 1, 2 and 3, see also Lemma 1.

In the sequel, in the simplified game, now machine 3 has become under-loaded and 2 optimum-loaded, while 1 remains over-loaded, see Figure 1b. In the same fashion as in the original game, `OpTop` assigns load o_3 to machine 3. Happily, the induced selfish scheduling of the remaining $r - o_3 - o_4 - o_5$ flow yields the overall optimum assignment for the system. That is, the remaining $r - o_3 - o_4 - o_5$ flow, when scheduled selfishly by the Followers, ignores machines 3, 4 and 5 (they assign no load to these machines) while their selfish behavior assigns induced Nash load $n'_i = o_i$ to each machine $i = 1, 2$, see Figure 1c.

In this example, algorithm `OpTop` needed to control a portion $\alpha_0 = \frac{o_3+o_4+o_5}{r}$, of the total flow r of jobs, in a way that the final induced load to each machine i equals its optimum value o_i, $i = 1, \ldots, 5$. `OpTop`'s objective is to impose the overall optimum by controlling the least possible portion α_0. The cornerstone for the stability of the load assigned by `OpTop` to any machine is Theorem 1. Intuitively, this theorem says that `OpTop` raises the latency of proper machines

Fig. 1. A dashed (or solid) line indicates the Nash (or Optimum) load n_i (or o_i) assigned to each machine $i = 1, \ldots, 5$. (a) Machines 1 and 2 (4 and 5) are over(under)-loaded while 3 is optimum-loaded. Then `OpTop` will assign load o_4 and o_5 to machines 4 and 5. (b) Now machines 4 and 5 received load o_4 and o_5 by `OpTop`. In the induced Nash assignment, machines 1 (3) become over(under)-loaded while 2 becomes optimum-loaded.(c) Finally, `OpTop` assigns load o_3 to machine 2. In the induced Nash assignment, machines 1 and 2 become optimum-loaded

sufficiently high, making them not appealing to selfish players, while retaining their respective optimum load.

Theorem 1. *Consider m machines with latency functions $\ell_j(x) = a_j x + b_j,\ j = 1, \ldots, m$. Let the Nash assignment $N = \langle n_1, \ldots, n_m \rangle$ of the total load r on the m machines. Suppose that for a Stackelberg strategy $S = \langle s_1, \ldots, s_m \rangle$ we have either $s_j \geq n_j$ or $s_j = 0, j = 1, \ldots, m$. Then for the induced Nash assignment $T = \langle t_1, \ldots, t_m \rangle$ of the remaining load $r - \sum_{i=1}^{m} s_i$ we have that $t_j = 0$ for each machine j such that $s_j \geq n_j,\ j = 1, \ldots, m$.*

Proof. By assigning load $s_j \geq n_j$ to machine j then for any induced load $t_j \geq 0$ to it, its latency is now increased up to $\widetilde{\ell}_j(t_j) = a_j t_j + \ell_j(s_j) \geq \ell_j(s_j) \geq \ell_j(n_j),\ j = 1, \ldots, m$. Since the induced Nash assignment T assigns total load $r - \sum_{i=1}^{m} s_i \leq \sum_{\{i: s_i = 0\}} n_i$, its is not now possible for any machine j with $s_j \geq n_j$ to become appealing to the selfish users, $j = 1, \ldots, m$.

Theorem 1 is valid for arbitrary increasing latency functions. Interestingly, a similar (monotonicity) argument can be found in [13]. Another difficulty for the evolution of the algorithm, is to describe the selfish play of the users in the remaining machines. To this end, Lemma 1 is helpful.

Lemma 1. *Let a set of machines $M = \{M_1, \ldots, M_m\}$ and the Nash assignment $N = \langle n_1, \ldots, n_m \rangle$ of the total load r on these machines. Fix a Stackelberg strategy $S = \langle s_1, \ldots, s_m \rangle$ such that either $s_j \geq n_j$ or $s_j = 0, j = 1, \ldots, m$. Then the initial game is equivalent to scheduling total flow: $r - \sum_{i=1}^{m} s_i$, to the subset $M' \subseteq M$ of machines: $M' = M \backslash \{M_i :\ s_i \geq n_i\},\ i = 1, \ldots, m$.*

Proof. It follows from Theorem 1.

Lemma 1 allows us to run recursively `OpTop` on the simplified game on the remaining machines. The empirical performance of `OpTop` is presented in Section 4, in Figures 3 and 5.

4 Experimental Validation of the Algorithms

All experiments presented in this section are performed using the package Matlab [29]. An extensive toolbox was created for manipulating large systems of machines for both linear and M/M/1 latency functions. All the routines of computing the Optimum and Nash assignments, the `LLF` and `NLS` strategies are also implemented in the same Toolbox [30].

Here we present results for 5-tuples of random machines for both linear and M/M/1 latency functions. Similar results were observed for k-tuples with $k \geq 5$. For total flow r, machine i receives a portion or flow x_i which incurs latency $\ell(x_i) = a_i x_i + b_i,\ i = 1, \ldots, 5$, where a_i, b_i are random numbers in $[0,\ r]$ and $\sum_{i=1}^{5} x_i = r$. The corresponding random M/M/1 latency functions are $\ell(x_i) = \frac{1}{u_i - x_i},\ i = 1, \ldots, 5$. We created many large collections of 5 tuples of machines,

where each such collection satisfies a predetermined value of the *parameter* $c \leq \frac{4}{3}$ (recall c is a lower bound of the price of anarchy value PoA). That is, for each generated random 5-tuple of machines, we compute the ratio of the cost of the Nash assignment to corresponding optimum one, and we store the 5-tuple to the appropriate c-valued collection of tuples. Intutively, a collection of tuples with particularly high value of c consists of highly competitive machines and the role of the Leader is important.

4.1 Linear Latency Functions

Comparing NLS to LLF. We know that LLF strategy induce a Follower assignment that drives the PoA to $\frac{4}{3+\alpha}$. We are interested in finding out how much better the NLS strategy does in comparison to LLF strategy. In other words we are interested in measuring the ratio $\frac{Cost\texttt{NLS}}{Cost\texttt{LLF}}$. The worst case would be when this ratio is 1, which means that the NLS strategy is the same as the LLF strategy. This phenomenon is expected since NLS is based on LLF but we are interested in finding out the how much similar is NLS to LLF. Based on intuition, we expected that in instances with higher values of PoA our strategy will do better than LLF. This will be the case with even lower α, since we may manipulate more machines in the subset M^* which is described in the pseudo code of NLS. This intuition was confirmed by the experiments, as it is shown in Figure 2. Both diagrams present the percentage of machines that had $\frac{Cost\texttt{NLS}}{Cost\texttt{LLF}} < 1$. What is remarkable is that NLS does better when the parameter c of the machine instances is increased from 1 up to 1.2. Then the corresponding portion of machines that had better performance using NLS is increased from 33% up to 62% of the instances.

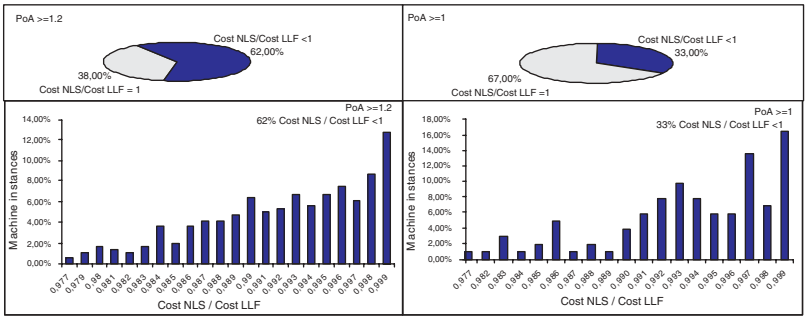

Fig. 2. Linear load functions: $\frac{CostNLS}{CostLLF}$ for $PoA \geq 1.2$ and for $PoA \geq 1$

We conjecture that the reason for this phenomenon is that systems with high PoA usually overload 1 or 2 machines, while the rest ones remain idle. Therefore, the αr flow assigned *local* optimally by the Leader to the subset of the idle machines remains unaffected.

Another interesting observation was that NLS does better than LLF for small α. For the instances with $PoA \geq 1$ the NLS strategy is better than LLF strategy

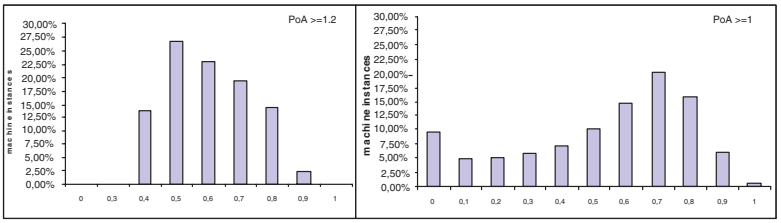

Fig. 3. Linear latency functions: The α_0 computed by `OpTop` to reach the overall optimum for $PoA \geq 1.2$ and for $PoA \geq 1$

for average $\alpha = 0.132$ while for instances with $PoA \geq 1.2$ the average α is higher and has the value α=0.313.

Finally, the average $\frac{Cost\texttt{NLS}}{Cost\texttt{LLF}}$ for $PoA \geq 1$ is 0.995 while the $\frac{Cost\texttt{NLS}}{Cost\texttt{LLF}}$ for $PoA \geq 1.2$ is 0.991.

Results for `OpTop`. The algorithm `OpTop` that we presented in Section 3, computes an upper bound to the amount αr of flow that the Leader must posses in order to induce optimal assignment. That is, we are interested in computing an upper bound to the *minimum flow* α_0 that the Leader must control to guarantee overall optimum performance induced by the Followers selfish play. In Figure 3, x-axis presents the portion α_0 needed to induce the overall optimum, while y-axis presents the corresponding percentage of 5-tuples of machines.

The results of our experiments on the set of machine instances are presented in Figure 3 below. In instances where $PoA \geq 1$ the portion α_0 of load flow the Leader has to control ranges in $\alpha_0 \in [0,\ 0.9]$ and its average value is $\alpha_0 = 0.5$.

Also in Figure 3, as PoA's lower bound increases up to 1.2, the range of α_0 the Leader has to control also increases, that is $\alpha_0 \in [0.4,\ 0.9]$. In this case its average value is $\alpha_0 = 0.6$.

4.2 Results for M/M/1 Latency Functions

For M/M/1 latency functions, (i.e. of the form $\frac{1}{u-x}$) the results are similar. The PoA of the systems with such load functions is not that different from the linear load functions. As we can see the `NLS` strategy does better for systems with an increased lower bound (parameter c) of PoA.

Once more, in Figure 4 we can see that `NLS` does better when the parameter c of the machine instances is increased from 1 up to 1.2. Then the corresponding portion of machines that had better performance using `NLS` is increased from 19% up to 43% of the instances. Furthermore, in the same figure, we see that the average $\frac{Cost\texttt{NLS}}{Cost\texttt{LLF}}$ for $PoA \geq 1$ is 0.992 while the $\frac{Cost\texttt{NLS}}{Cost\texttt{LLF}}$ for $PoA \geq 1.2$ is 0.988.

The results of our experiments for `OpTop` on the set of machine instances are presented in Figure 5 below. In instances where $PoA \geq 1$ the portion α_0 of flow the Leader has to control to induce the overall optimum ranges in $\alpha_0 \in [0.2,\ 0.9]$ and its average value is $\alpha_0 = 0.57$. Also in this figure, as PoA's lower bound increases up to 1.2, the range of α_0 the Leader has to control is in $\alpha_0 \in [0.2,\ 1]$.

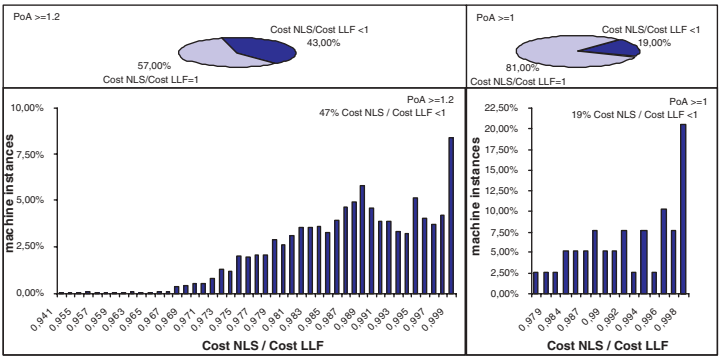

Fig. 4. M/M/1 latency functions: $\frac{CostNLS}{CostLLF}$ for $PoA \geq 1.2$ and for $PoA \geq 1$

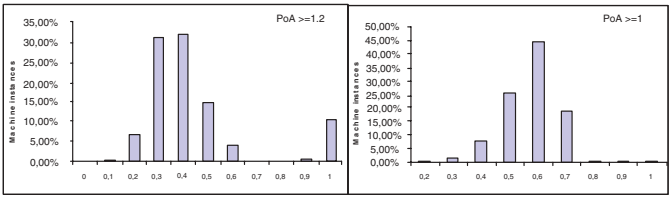

Fig. 5. M/M/1 latency functions: The α_0 computed by `OpTop` to reach the overall optimum for $PoA \geq 1.2$ and for $PoA \geq 1$

Rather unexpectedly, in this case its average value has been reduced to $\alpha_0 = 0.44$. Further work will focus on machine instances with arbitrary latency functions, where the PoA is greater or even unbounded and the results are expected to be more interesting than those of the linear load functions and M/M/1 functions.

Acknowledgements

We thank the anonymous referees for their comments that substantially improved the presentation of our ideas.

References

1. V. S. Anil Kumar, Madhav V. Marathe. Improved Results for Stackelberg Scheduling Strategies. *In Proc. ICALP '02*, pp. 776-787.
2. T. Basar, G. J. Olsder.*Dynamic Noncooperative Game Theory.* SIAM, 1999.
3. A. Czumaj and B. Voecking. Tight bounds for worst-case equilibria. *In Proc. SODA '02.*
4. A. Czumaj. *Selfish Routing on the Internet*, Handbook of Scheduling: Algorithms, Models, and Performance Analysis,CRC Press, 2004.

5. J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *In Proc. STOC '00.*
6. D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, P. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. *In Proc. ICALP '02*, pp 123–134.
7. P. Golle, K. Leyton-Brown, I. Mironov, M. Lillibridge. Incentives for Sharing in Peer-to-Peer Networks.Full version, *In Proc. WELCOM 01.*
8. S. A. Grandhi, R. D. Yates and D. J. Goodman: Resource Allocation for Cellular Radio Systems, *In Proc. IEEE Transactions on Vehicular Technology*, vol. 46, no. 3, pp. 581-587, August 1997.
9. Y.A. Korilis, A.A. Lazar, A. Orda: Achieving network optima using stackelberg routing strategies, *In Proc. IEEE/ACM Transactions of Networking*, 1997.
10. Y. A. Korilis, A. A. Lazar and A. Orda. The designer's perspective to noncooperative networks. *In Proc. IEEE INFOCOM 95.*
11. Y.A. Korilis, A.A. Lazar, A. Orda: Capacity allocation under noncooperative routing, *In Proc. IEEE/Transactions on Automatic Control*, 1997.
12. E. Koutsoupias and C. Papadimitriou. Worst-case Equilibria. *In Proc. STACS '99*, pp. 387–396.
13. H. Lin, T. Roughgarden, and E. Tardos, A Stronger Bound on Braess's Paradox, SODA 2004.
14. M. Mavronikolas and P. Spirakis: The price of Selfish Routing, *In Proc. STOC' 01.*
15. R. B. Myerson. *Game Theory: Analysis of Conflict.* Harvard University Press, 1991
16. N. Nisan and A. Ronen. Algorithmic mechanism design.*In Proc. STOC'99*, pp 129-140
17. N. Nisan. Algorithms for selfish agents: Mechanism design for distributed computation. *In Proc. STACS '99.*
18. M. J. Osborne, A. Rubinstein. *A course in Game Theory*, MIT Press
19. G. Owen. *Game Theory.* Academic Press. Orlando, FL, 3rd Edition, 1995.
20. C. Papadimitriou. Game Theory and Mathematical Economics: A Theoratical Computer Scientist's Introduction. *In Proc. FOCS '01.*
21. C. Papadimitriou. Algorithms, Games, and the Internet. *In Proc. STOC '01*, pp 749-753.
22. T. Roughgarden. The price of anarchy is independent of the network topology. *In Proc. STOC '02*, pp 428-437.
23. T. Roughgarden. Stackelberg scheduling strategies. *In Proc. STOC '01*, pp 104-113.
24. T. Roughgarden, E. Tardos. How bad is Selfish Routing?. *In Proc. FOCS '00*,pp 93-102.
25. T. Roughgarden. Designing networks for selfish users is hard. *In Proc. FOCS '01*, pp 472–481.
26. H. von Stackelberg. *Marktform aund Gleichgewicht.* Spiringer-Verlag, 1934
27. H. Yaiche, R. Mazumdar and C. Rosenberg. Distributed algorithms for fair bandwidth allocation in broadband networks, *In Proc. INFOCOM 00.*
28. H. Yaiche , R. Mazumdar and C. Rosenberg. A game-theoretic framework for bandwidth allocation and pricing in broadband networks, *In Proc. the IEEE/ACM Transactions on Networking*, Vol. 8, No. 5, Oct. 2000, pp. 667-678.
29. *Optimization Toolbox for use with MATLAB*, User's Guide, MathWorks.
30. http://students.ceid.upatras.gr/∼ politop/stackTop, Stackelberg Strategies Toolbox.