

A Game-Theoretic Model and Algorithm for Load Balancing in Distributed Systems *

Daniel Grosu and Anthony T. Chronopoulos
Department of Computer Science,
University of Texas at San Antonio,
6900 N. Loop 1604 West, San Antonio, TX 78249
{dgrosu, atc}@cs.utsa.edu

Abstract

In this paper we present a game theoretic framework for obtaining a user-optimal load balancing scheme in heterogeneous distributed systems. We formulate the static load balancing problem in heterogeneous distributed systems as a noncooperative game among users. For the proposed noncooperative load balancing game, we present the structure of the Nash equilibrium. Based on this structure we derive a new distributed load balancing algorithm. Finally, the performance of our noncooperative load balancing scheme is compared with that of other existing schemes. Our scheme guarantees the optimality of allocation for each user in the distributed system.

1 Introduction

A distributed system can be viewed as a collection of computing and communication resources shared by active users. When the demand for computing power increases the load balancing problem becomes important. A general formulation of this problem is as follows: given a large number of jobs, find the allocation of jobs to computers optimizing a given objective function (e.g. total execution time).

There are three typical approaches to load balancing problem in distributed systems:

1. *Global approach*: In this case there is only one decision maker that optimizes the response time of the entire system over all jobs and the operating point is called *social (overall) optimum*. This is the classical approach and has been studied extensively using different techniques such as

nonlinear optimization [7, 8, 12, 18, 19] and polymatroid optimization [15].

2. *Cooperative approach*: In this case there are several decision makers (e.g. jobs, computers) that cooperate in making the decisions such that each of them will operate at its optimum. Decision makers have complete freedom of preplay communication to make joint agreements about their operating points. This situation can be modeled as a cooperative game and game theory offers a suitable modeling framework [2].

3. *Noncooperative approach*: In this case there are several decision makers (e.g. users, jobs) that are not allowed to cooperate in making decisions. Each decision maker optimizes its own response time independently of the others and they all eventually reach an equilibrium. This situation can be viewed as a noncooperative game among decision makers. The equilibrium is called *Nash equilibrium* [2] and it can be obtained by a distributed noncooperative policy. At the Nash equilibrium a decision maker cannot receive any further benefit by changing its own decision. If the number of decision makers is not finite the Nash equilibrium is called *Wardrop equilibrium* [6].

Past results

There exist only few studies on game theoretic models and algorithms for load balancing in distributed systems. Kameda *et al.* [6] studied noncooperative games and derived load balancing algorithms for computing the Wardrop equilibrium in single class and multi-class job distributed systems. Roughgarden [16] formulated the load balancing problem as a Stackelberg game. In this type of noncooperative game one player acts as a leader and the rest as followers. He showed that it is NP-hard to compute the optimal Stackelberg strategy and presents efficient algorithms to compute strategies inducing near-optimal solutions.

Routing traffic in networks is a closely related problem that received more attention. Orda *et al.* [14] studied a noncooperative game in a network of parallel links with convex

*This research was supported, in part, by research grants from: (1) NASA NAG 2-1383 (1999-2001); (2) State of Texas Higher Education Coordinating Board through the Texas Advanced Research/Advanced Technology Program ATP 003658-0442-1999. Some reviewer comments helped enhance the quality of presentation.

cost functions. They studied the existence and uniqueness of the Nash equilibrium. Altman *et al.* [1] investigated the same problem in a network of parallel links with linear cost functions. Korilis *et al.* [10] considered the capacity allocation problem in a network shared by noncooperative users. They studied the structure and the properties of Nash equilibrium for a routing game with M/M/1 type cost functions. An important line of research was initiated by Koutsoupias and Papadimitriou [11], who considered a noncooperative routing game and proposed the ratio between the worst possible Nash equilibrium and the overall optimum as a measure of effectiveness of the system. Roughgarden and Tardos [17] showed that in a network in which the link cost functions are linear the flow at Nash equilibrium has total latency at most 4/3 that of the overall optimal flow. They also showed that if the link cost functions are assumed to be only continuous and nondecreasing the total latency may be arbitrarily larger than the minimum possible total latency.

Our results

Most of the previous studies on static load balancing considered as their main objective the minimization of overall expected response time. This is difficult to achieve in distributed systems where there is no central authority controlling the allocation and users are free to act in a selfish manner. Our goal is to find a formal framework for characterizing user-optimal allocation schemes in distributed systems. The framework was provided by noncooperative game theory which has been applied to routing and flow control problems in networks but not to load balancing in distributed systems. Using this framework we formulate the load balancing problem in distributed systems as a noncooperative game among users. The Nash equilibrium provides a user-optimal operation point for the distributed system. We give a characterization of the Nash equilibrium and a distributed algorithm for computing it. We compare the performance of our noncooperative load balancing scheme with that of other existing schemes. Our scheme guarantees the optimality of allocation for each user in the distributed system.

Organization

The paper is structured as follows. In Section 2 we present the system model and we introduce our load balancing noncooperative game. In Section 3 we derive a greedy distributed algorithm for computing the Nash equilibrium for our load balancing game. In Section 4 the performance of our load balancing scheme is compared with those of other existing schemes. In Section 5 we draw conclusions and present future directions.

2 Load balancing as a noncooperative game among users

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

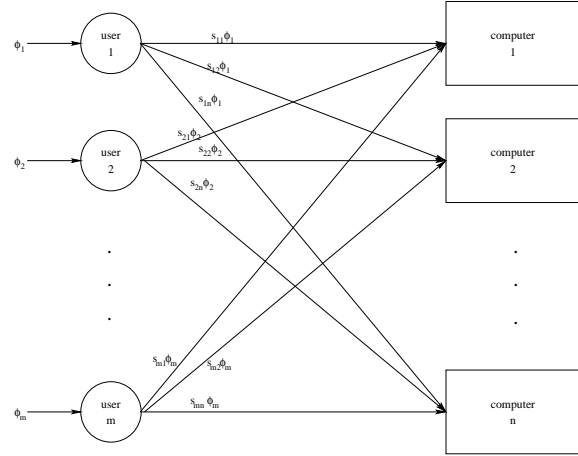


Figure 1. The distributed system model.

modeled as an M/M/1 queueing system (i.e. Poisson arrivals and exponentially distributed processing times) [9]. Computer i is characterized by its average processing rate μ_i , $i = 1, \dots, n$. Jobs are generated by user j with an average rate ϕ_j , and $\Phi = \sum_{j=1}^m \phi_j$ is the total job arrival rate in the system. The total job arrival rate Φ must be less than the aggregate processing rate of the system (i.e. $\Phi < \sum_{i=1}^n \mu_i$). The system model is presented in Figure 1. The users have to decide on how to distribute their jobs to computers such that they will operate optimally. Thus user j ($j = 1, \dots, m$) must find the fraction s_{ji} of all its jobs that are assigned to computer i ($\sum_{i=1}^n s_{ji} = 1$ and $0 \leq s_{ji} \leq 1$, $i = 1, \dots, n$) such that the expected execution time of its jobs is minimized.

We formulate this problem as a noncooperative game among users under the assumption that users are 'selfish'. This means that they minimize the expected response time of their own jobs.

Let s_{ji} be the fraction of jobs that user j sends to computer i . The vector $\mathbf{s}_j = (s_{j1}, s_{j2}, \dots, s_{jn})$ is called the *load balancing strategy* of user j . The vector $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_m)$ is called the *strategy profile* of the load balancing game.

We assume that each computer is modeled as an M/M/1 queueing system and the expected response time at computer i is given by:

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

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

$$D_j(\mathbf{s}) = \sum_{i=1}^n s_{ji} F_i(\mathbf{s}) = \sum_{i=1}^n \frac{s_{ji}}{\mu_i - \sum_{k=1}^m s_{ki} \phi_k} \tag{2}$$

The goal of user j is to find a feasible load balancing strategy \mathbf{s}_j such that $D_j(\mathbf{s})$ is minimized.

A feasible load balancing strategy profile \mathbf{s} must satisfy the following restrictions:

- (i) Positivity: $s_{ji} \geq 0, i = 1, \dots, n, j = 1, \dots, m;$
- (ii) Conservation: $\sum_{i=1}^n s_{ji} = 1, j = 1, \dots, m;$
- (iii) Stability: $\sum_{j=1}^m s_{ji} \phi_j < \mu_i, i = 1, \dots, n;$

The decision of user j depends on the load balancing decisions of other users since $D_j(\mathbf{s})$ is a function of \mathbf{s} . We adopt here the Nash equilibrium as optimality concept [2].

Definition 2.1 A Nash equilibrium of the load balancing game defined above is a strategy profile \mathbf{s} such that for every user j :

$$\mathbf{s}_j \in \arg \min_{\tilde{\mathbf{s}}_j} D_j(\mathbf{s}_1, \dots, \tilde{\mathbf{s}}_j, \dots, \mathbf{s}_m) \quad (3)$$

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.

Similar games were studied in the context of control flow and routing in networks. Orda *et al.* [14] proved that if the expected response time functions are continuous, convex and increasing there exist a unique Nash equilibrium for the game. The closest work to our study is that of Korilis *et al.* [10] in which it is studied a similar game in the context of capacity allocation in networks. They studied the structure and properties of Nash equilibrium for the game. Here, we are interested in finding a way to compute the Nash equilibrium for our load balancing noncooperative game.

We need to determine the strategy profile of user j which must be optimal with respect to the other users strategies. Let $\mu_i^j = \mu_i - \sum_{k=1, k \neq j}^m s_{ki} \phi_k$ be the *available processing rate* at processor i as seen by user j . The problem of computing the optimal strategy of user j ($j = 1, \dots, m$) reduces to computing the optimal strategy for a system with n processors having μ_i^j ($i = 1, \dots, n$) as processing rates and ϕ_j as the job arrival rate in the system.

For user j , the associated optimization problem (OP _{j}) can be described as follows:

$$\min_{\mathbf{s}_j} D_j(\mathbf{s}) \quad (4)$$

subject to the constraints:

$$s_{ji} \geq 0, \quad i = 1, \dots, n \quad (5)$$

$$\sum_{i=1}^n s_{ji} = 1 \quad (6)$$

$$\sum_{k=1}^m s_{ki} \phi_k < \mu_i, \quad i = 1, \dots, n \quad (7)$$

Remark: The strategies of all the other users are kept fixed, thus the variables involved in OP _{j} are the load fractions of user j , i.e. $\mathbf{s}_j = (s_{j1}, s_{j2}, \dots, s_{jn})$.

There exist few algorithms for finding the optimum for similar optimization problems. One was proposed by Tantawi and Towsley [19] but it is complex and involves a method for solving a nonlinear equation. Another one which is very close to our approach was proposed by Tang and Chanson [18]. These algorithms cannot be applied directly for our problem because there are some other issues that must be addressed. Our approach is inspired by [18] but it considers a different model in which the influence of the other users' decisions on the optimum is taken into account. We propose in the following an algorithm for computing the optimum considering our model.

We can characterize the optimal solution of OP _{j} as follows:

Theorem 2.1 Assuming that computers are ordered in decreasing order of their available processing rates ($\mu_1^j \geq \mu_2^j \geq \dots \geq \mu_n^j$), the solution \mathbf{s}_j of the optimization problem OP _{j} is given by:

$$s_{ji} = \begin{cases} \frac{1}{\phi_j} \left(\mu_i^j - \sqrt{\mu_i^j \frac{\sum_{i=1}^{c_j} \mu_i^j - \phi_j}{\sum_{i=1}^{c_j} \sqrt{\mu_i^j}}} \right) & \text{if } 1 \leq i < c_j \\ 0 & \text{if } c_j \leq i \leq n \end{cases} \quad (8)$$

where c_j is the minimum index that satisfies the inequality:

$$\sqrt{\mu_{c_j}^j} \leq \frac{\sum_{k=1}^{c_j} \mu_k^j - \phi_j}{\sum_{k=1}^{c_j} \sqrt{\mu_k^j}} \quad (9)$$

Proof: In Appendix.

Based on the above theorem we derived the following algorithm for solving user j 's optimization problem OP _{j} .

OPTIMAL($\mu_1^j, \dots, \mu_n^j, \phi_j$)

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

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

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

The following theorem proves the correctness of this algorithm.

Theorem 2.2 The load balancing strategy ($s_{j1}, s_{j2}, \dots, s_{jn}$) computed by the OPTIMAL algorithm solves the optimization problem OP _{j} and is the optimal strategy for user j .

Proof: In Appendix.

Remarks: (1) The execution time of this algorithm is $O(n \log n)$. This is due to the sorting procedure in step 1. (2) To execute this algorithm each user needs to know the available processing rate at each computer and its job arrival rate. The available processing rate can be determined by statistical estimation of the run queue length of each processor.

3 A distributed load balancing algorithm

The computation of Nash equilibrium might require some coordination between the users. From the practical point of view we need decentralization and this can be obtained by using distributed greedy best response algorithms [2]. In these algorithms each user updates from time to time its load balancing strategy by computing the best response against the existing load balancing strategies of the other users. In our case each user will update its strategy in a round-robin fashion.

Based on the OPTIMAL algorithm presented in the previous section, we devised the following greedy best reply algorithm for computing the Nash equilibrium for our non-cooperative load balancing game.

We use the following notations: l - the number of iterations; $\mathbf{s}_j^{(l)}$ - 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; **Send**(j , (p , q)) - send the message (p , q) to user j ; **Recv**(j , (p , q)) - receive the message (p , q) from user j .

NASH distributed load balancing algorithm:

User j , ($j = 1, \dots, m$) executes:

1. Initialization:
 - $\mathbf{s}_j^{(0)} \leftarrow \mathbf{0}$; $\mathbf{D}_j^{(0)} \leftarrow \mathbf{0}$; $l \leftarrow 1$;
2. **while** (1) **do**
 - if**($j \neq 1$)
 - Recv**($j - 1$, ($norm$, l));
 - if** ($norm = -1$)
 - if** ($j \neq n$) **Send**($j + 1$, (-1 , -1));
 - exit**;
 - else**
 - if** ($l \neq 1$)
 - Recv**(n , ($norm$, l));
 - if** ($norm < \epsilon$)
 - Send**($j + 1$, (-1 , -1));
 - exit**;
 - if**($j = 1$) $norm \leftarrow 0$;
 - for** $i = 1, \dots, n$ **do**
 - Obtain μ_i^j by inspecting the run queue of each computer: ($\mu_i^j \leftarrow \mu_i - \sum_{k=1, k \neq j}^m s_{ki} \phi_k$);
 - $\mathbf{s}_j^{(l)} \leftarrow \text{OPTIMAL}(\mu_1^j, \dots, \mu_n^j, \phi_j)$;
 - Compute $D_j^{(l)}$;
 - $normj \leftarrow |D_j^{(l-1)} - D_j^{(l)}|$;

```

norm ← norm + normj;
if( $j = 1$ )  $l \leftarrow l + 1$ ;
if( $j = n$ ) Send(1, ( $norm$ ,  $l$ ));
else Send( $j + 1$ , ( $norm$ ,  $l$ ));

```

The execution of this algorithm is initiated 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.

An important practical question is whether such 'best reply' algorithms converge to the Nash equilibrium. The only known 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 [14] or by $m \geq 2$ users but with linear cost links [1]. For M/M/1 type cost functions there is no known proof that such algorithms converge for more than two users. Several experiments done on different settings show that they converge for more than two users. In the next section we present experiments that confirm this hypothesis. The convergence proof for more than two users is still an open problem.

4 Experimental results

4.1 Simulation environment

The simulations were carried out using Sim++ [4], a simulation software package written in C++. This package provides an application programming interface which allows the programmer to call several functions related to event scheduling, queueing, preemption and random number generation. 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 FCFS (first-come-first-served) order. Each computer is modeled as an M/M/1 queueing system [9]. The main performance metrics used in our simulations are the *expected response time* and the *fairness index*. The *fairness index* $I(\mathbf{D}) = \frac{[\sum_{j=1}^m D_j]^2}{n \sum_{j=1}^m D_j^2}$,

was proposed in [5] to quantify the fairness of load balancing schemes. Here the parameter \mathbf{D} is the vector $\mathbf{D} = (D_1, D_2, \dots, D_m)$ where D_j is the expected execution time of user j 's jobs. The simulations were run over several thousands of seconds, sufficient to generate a total of 1 to 2 millions jobs typically. Each run was replicated five times with different random number streams and the results averaged over replications. The standard error is less than 5% at the 95% confidence level.

4.2 Performance evaluation

For comparison purposes we consider three existing static load balancing schemes [3, 6, 7]. A brief description of these schemes is given below:

- **Proportional Scheme (PS)** [3]: 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. It can be shown that for this scheme the fairness index is always 1.

- **Global Optimal Scheme (GOS)** [7]: This scheme minimizes the expected execution time over all jobs executed by the system. The load fractions (\mathbf{s}) are obtained by solving the nonlinear optimization problem $\min_{\mathbf{s}} \frac{1}{\Phi} \sum_{j=1}^m \phi_j D_j(\mathbf{s})$ subject to the constraints (5) - (7). This scheme provides the overall optimum for the expected execution time but it is not user-optimal and is unfair.

- **Individual Optimal Scheme (IOS)** [6]: In this scheme each job optimizes its response time for itself independently of others. In general the Wardrop equilibrium, which is the solution given by this scheme, is not optimal and in some cases we expect worse response time than the other policies [6]. It is based on an iterative procedure that is not very efficient. For a complete description of IOS algorithm see [6]. The advantage of this scheme is that it provides a fair allocation.

Remark: Among the three schemes described above, the IOS scheme is the only scheme that is based on game theoretic concepts.

We evaluated the schemes presented above under various system loads and configurations. Also the convergence of the NASH load balancing algorithm is investigated. In the following we present and discuss the simulation results.

4.2.1 The convergence of NASH algorithm

An important issue related to the greedy best reply algorithm presented above is the dynamics of reaching the equilibrium. We consider first the NASH algorithm using $\mathbf{s}^{(0)} = \mathbf{0}$ as the initialization step. This variant of the algorithm will be called NASH_0. This initialization step is an obvious choice but it may not lead to a fast convergence to the equilibrium.

We propose a variant of the algorithm in which the initialization step is replaced by:

1. Initialization:

$$\mathbf{D}_j^{(0)} \leftarrow \mathbf{0}; l \leftarrow 1;$$
for $i = 1, \dots, n$ **do**

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

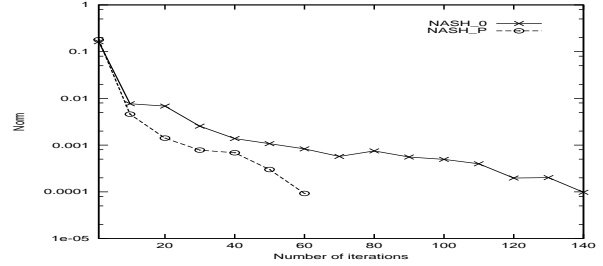


Figure 2. Norm vs. number of iterations.

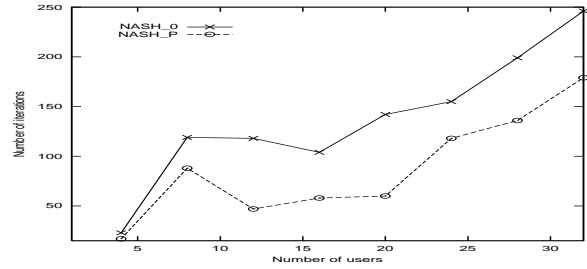


Figure 3. Convergence of best reply algorithms.

We call this new version NASH_P. 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. To study the convergence of these algorithms we consider a systems with 16 computers shared by 10 users. The norm vs. the number of iterations is shown in Figure 2. It can be seen that 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 to more than a half compared with NASH_0.

Next, we study the influence of the number of users on the convergence of both algorithms. In Figure 3 we present the number of iterations needed to reach the equilibrium for a system with 16 computers and a variable number of users (from 4 to 32). It can be observed that NASH_P significantly outperforms NASH_0 reducing the number of iterations needed to reach the equilibrium in all the cases.

4.2.2 Effect of system utilization

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

users. In Table 1, we present the system configuration. The first row contains the relative processing rates of each of the four computer types. Here, the relative processing rate for computer C_i is defined as the ratio of the processing rate of C_i to the processing rate of the slowest computer in the system. The second row contains the number of computers in the system corresponding to each computer type. The last row shows the processing rate of each computer type in the system. We consider only computers that are at most ten times faster than the slowest because this is the case in most of the current heterogeneous distributed systems.

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

Table 1. System configuration.

In Figure 4, we present the expected response time of the system and the fairness index for different values of system utilization (ranging from 10% to 90%). *System utilization* (ρ) is defined as the ratio of total arrival rate to aggregate processing rate of the system: $\rho = \frac{\Phi}{\sum_{i=1}^n \mu_i}$.

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

An important performance metric is the fairness index. The PS and IOS schemes maintain a fairness index of 1 over the whole range of system loads. It can be shown that the PS has a fairness index of 1 which is a constant independent of the system load. The fairness index of GOS varies from 1 at low load, to 0.92 at high 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 (i.e. it is user-optimal). User-optimality and decentralization are the main advantages of NASH scheme.

An interesting issue is the impact of static load balancing schemes on individual users. In Figure 5, we present the expected response time for each user considering all static schemes at medium load ($\rho=60\%$). The PS and IOS schemes guarantee equal expected response times for all users but with the disadvantage of a higher expected execution time for their jobs. It can be observed that in the case of GOS scheme there are large differences in users' expected

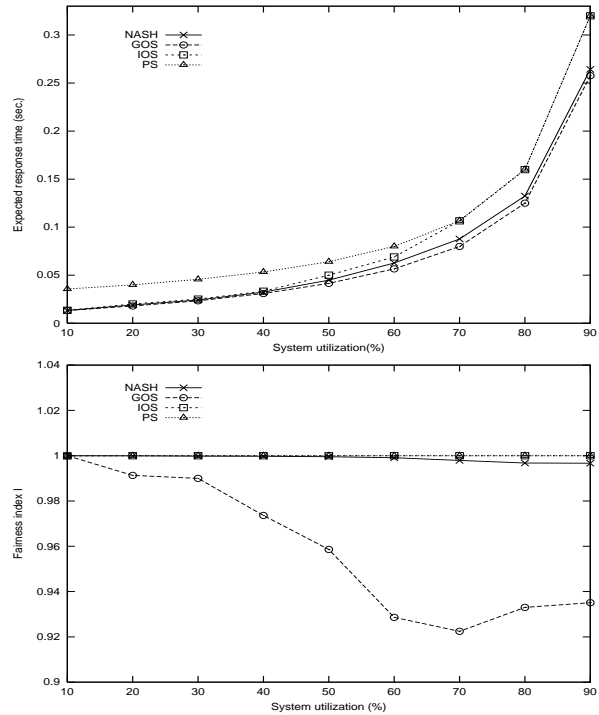


Figure 4. The expected response time and fairness index vs. system utilization.

execution times. NASH scheme provides the minimum possible expected execution time for each user. Thus, from users' perspective NASH is the most desirable scheme.

4.2.3 Effect of heterogeneity

In a distributed system, heterogeneity usually consists of: processor speed, memory and I/O. A simple way to characterize system heterogeneity is to use the processor speed. Furthermore, it is reasonable to assume that a computer with high speed processor will have matching resources (memory and I/O). One of the common measures of heterogeneity is the *speed skewness* [18] which is defined as the ratio of maximum processing rate to minimum processing rate of the computers. This measure is somehow limited but for our goals it is satisfactory.

In this section, we investigate the effectiveness of load balancing schemes by varying the speed skewness. We simulate a system of 16 heterogeneous computers: 2 fast and 14 slow. The slow computers have a relative processing rate of 1 and we varied the relative processing rate of the fast computers from 1 (which correspond to a homogeneous system) to 20 (which correspond to a highly heterogeneous system). The system utilization was kept constant $\rho = 60\%$.

In Figure 6, we present the effect of speed skewness

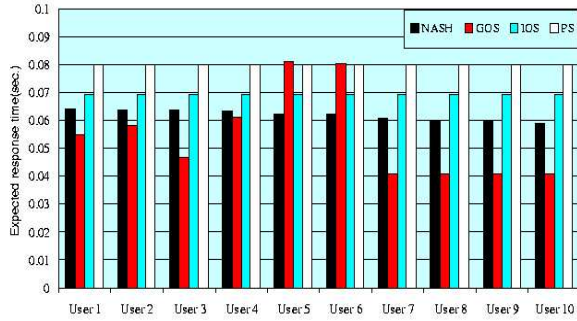


Figure 5. Expected response time for each user.

on the expected response time and fairness. It can be observed that increasing the speed skewness the GOS and NASH schemes yield almost the same expected response time which means that in highly heterogeneous systems the NASH scheme is very effective. NASH scheme has the additional advantage of decentralization and user-optimality which is very important in actual distributed systems. PS scheme performs poorly because it overloads the slowest computers. The IOS scheme performs well at high speed skewness approaching the performance of NASH and GOS, but at low speed skewness it performs poorly.

5 Conclusion

In this paper we have presented a game theoretic framework for obtaining a user-optimal load balancing scheme in heterogeneous distributed systems. We formulated the load balancing problem in heterogeneous distributed systems as a noncooperative game among users. For this game the Nash equilibrium provides an user-optimal operation point for the distributed system. For the proposed noncooperative load balancing game, we presented the structure of the Nash equilibrium. Based on this structure we derived a new distributed algorithm for computing it. We compared the performance of our noncooperative load balancing scheme with other existing schemes. Future work will address the development of game theoretic models for load balancing in the context of uncertainty as well as game theoretic models for dynamic load balancing.

References

[1] E. Altman, T. Basar, T. Jimenez, and N. Shimkin. Routing in two parallel links: Game-theoretic distributed algorithms. *to appear in: J. of Parallel and Distributed Computing*, 2001.
 [2] T. Basar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. SIAM, 1998.

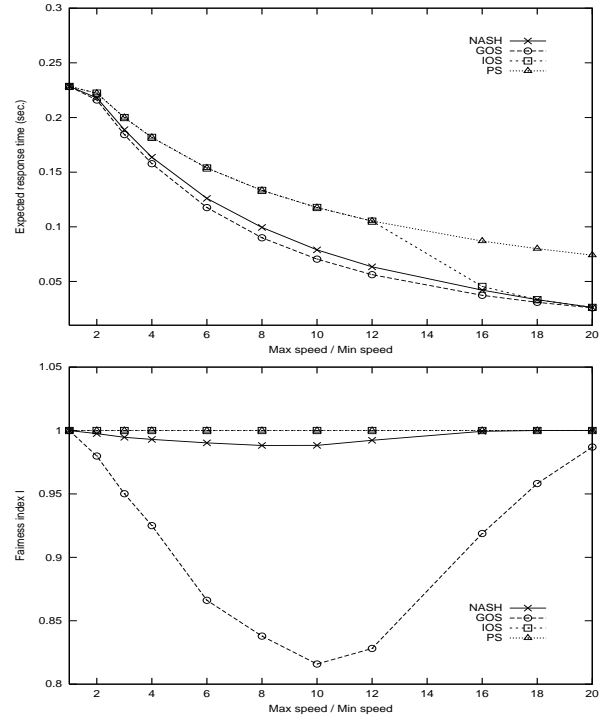


Figure 6. The effect of heterogeneity on the expected response time and fairness index.

[3] Y. C. Chow and W. H. Kohler. Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Trans. Comput.*, C-28(5):354–361, May 1979.
 [4] R. M. Cubert and P. Fishwick. *Sim++ Reference Manual*. CISE, Univ. of Florida, July 1995.
 [5] R. K. Jain, D.M. Chiu, and W. R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Technical Report DEC-TR-301, Digital Equipment Corporation, Eastern Research Lab, 1984.
 [6] H. Kameda, J. Li, C. Kim, and Y. Zhang. *Optimal Load Balancing in Distributed Computer Systems*. Springer Verlag, London, 1997.
 [7] C. Kim and H. Kameda. Optimal static load balancing of multi-class jobs in a distributed computer system. In *Proc. of the 10th Intl. Conf. on Distributed Computing Systems*, pages 562–569, May 1990.
 [8] C. Kim and H. Kameda. An algorithm for optimal static load balancing in distributed computer systems. *IEEE Trans. Comput.*, 41(3):381–384, March 1992.
 [9] L. Kleinrock. *Queueing Systems - Volume 1: Theory*. John Wiley and Sons, 1975.
 [10] Y. A. Korilis, A. A. Lazar, and A. Orda. Capacity allocation under noncooperative routing. *IEEE Trans. Automatic Control*, 42(3):309–325, March 1997.

- [11] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proc. of the 16th Annual Symp. on Theoretical Aspects of Computer Science*, pages 404–413, 1999.
- [12] J. Li and H. Kameda. Load balancing problems for multiclass jobs in distributed/parallel computer systems. *IEEE Trans. Comput.*, 47(3):322–332, March 1998.
- [13] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, Mass., 1984.
- [14] A. Orda, R. Rom, and N. Shimkin. Competitive routing in multiuser communication networks. *IEEE/ACM Trans. Networking*, 1(5):510–521, October 1993.
- [15] K. W. Ross and D. D. Yao. Optimal load balancing and scheduling in a distributed computer system. *Journal of the ACM*, 38(3):676–690, July 1991.
- [16] T. Roughgarden. Stackelberg scheduling strategies. In *Proc. of the 33rd Annual ACM Symp. on Theory of Computing (STOC'2001)*, pages 104–113, July 2001.
- [17] T. Roughgarden and E. Tardos. How bad is selfish routing? In *Proc. of the 41th IEEE Symp. on Foundations of Computer Science*, pages 93–102, November 2000.
- [18] X. Tang and S. T. Chanson. Optimizing static job scheduling in a network of heterogeneous computers. In *Proc. of the Intl. Conf. on Parallel Processing*, pages 373–382, August 2000.
- [19] A. N. Tantawi and D. Towsley. Optimal static load balancing in distributed computer systems. *Journal of the ACM*, 32(2):445–465, April 1985.

A Appendix

Proof of Theorem 2.1:

We begin with the observation that at the Nash equilibrium the stability condition (7) is always satisfied because of (3) and the fact that the total arrival rate (Φ) does not exceed the total processing rate of the distributed system. Thus we consider OP_j problem with only two restrictions, (5) and (6). We first show that $D_j(\mathbf{s})$ is a convex function in \mathbf{s}_j and that the set of feasible solutions defined by the constraints (5) and (6) is convex. From (2) it can be easily show that $\frac{\partial D_j(\mathbf{s})}{\partial s_{ji}} \geq 0$ and $\frac{\partial^2 D_j(\mathbf{s})}{\partial (s_{ji})^2} \geq 0$ for $i = 1, \dots, n$. This means that the Hessian of $D_j(\mathbf{s})$ is positive which implies that $D_j(\mathbf{s})$ is a convex function of the load fractions \mathbf{s}_j . The constraints are all linear and they define a convex polyhedron. Thus, OP_j involves minimizing a convex function over a convex feasible region and the first order Kuhn-Tucker conditions are necessary and sufficient for optimality [13].

Let $\alpha \geq 0, \eta_i \geq 0, i = 1, \dots, n$ denote the Lagrange multipliers [13]. The Lagrangian is:

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

The Kuhn-Tucker conditions imply that $s_{ji}, i = 1, \dots, n$ is the optimal solution to OP_j if and only if there exists $\alpha \geq 0, \eta_i \geq 0, i = 1, \dots, n$ such that: $\frac{\partial L}{\partial s_{ji}} = 0; \frac{\partial L}{\partial \alpha} = 0; \eta_i s_{ji} = 0, \eta_i \geq 0, s_{ji} \geq 0, i = 1, \dots, n$. These conditions are equivalent to:

$$\alpha = \frac{\mu_i^j}{(\mu_i^j - s_{ji}\phi_j)^2}, \quad \text{if } s_{ji} > 0 \quad 1 \leq i \leq n \quad (10)$$

$$\alpha \leq \frac{\mu_i^j}{(\mu_i^j - s_{ji}\phi_j)^2}, \quad \text{if } s_{ji} = 0 \quad 1 \leq i \leq n \quad (11)$$

$$\sum_{i=1}^n s_{ji} = 1, \quad s_{ji} \geq 0, \quad i = 1, \dots, n \quad (12)$$

Claim: Obviously, a computer with a higher average processing rate should have a higher fraction of jobs assigned to it. Under the assumption on the ordering of computers ($\mu_1^j \geq \mu_2^j \geq \dots \geq \mu_n^j$), we have the following order on load fractions: $s_{j1} \geq s_{j2} \geq \dots \geq s_{jn}$. This implies that may exist situations in which the slow computers have no jobs assigned to them. This means that there exist an index c_j ($1 \leq c_j \leq n$) so that $s_{ji} = 0$ for $i = c_j, \dots, n$.

From (10) and based on the above claims we can obtain by summation the following equation:

$$\sum_{i=1}^{c_j-1} \sqrt{\mu_i^j} = \sqrt{\alpha} \left(\sum_{i=1}^{c_j-1} \mu_i^j - \sum_{i=1}^{c_j-1} s_{ji}\phi_j \right) \quad (13)$$

Using (11) the above equation becomes:

$$\sqrt{\alpha} = \frac{\sum_{i=1}^{c_j-1} \sqrt{\mu_i^j}}{\sum_{i=1}^{c_j-1} \mu_i^j - \sum_{i=1}^{c_j-1} s_{ji}\phi_j} \leq \frac{1}{\sqrt{\mu_{c_j}^j}} \quad (14)$$

This is equivalent to:

$$\sqrt{\mu_{c_j}^j} \sum_{i=1}^{c_j} \sqrt{\mu_i^j} \leq \sum_{i=1}^{c_j} \mu_i^j - \phi_j \quad (15)$$

Thus, the index c_j is the minimum index that satisfies the above equation and the result follows. \square

Proof of Theorem 2.2:

The while loop in step 3 finds the minimum index c_j for which $\sqrt{\mu_{c_j}^j} \leq \frac{\sum_{k=1}^{c_j} \mu_k^j - \phi_j}{\sum_{k=1}^{c_j} \sqrt{\mu_k^j}}$. In the same loop, s_{ji} are set to zero for $i = c_j, \dots, n$. In step 4, s_{ji} is set equal to $\frac{1}{\phi_j} \left(\mu_i^j - \sqrt{\mu_i^j} \frac{\sum_{i=1}^{c_j} \mu_i^j - \phi_j}{\sum_{i=1}^{c_j} \sqrt{\mu_i^j}} \right)$ for $i = 1, \dots, c_j - 1$. These are in accordance with Theorem 2.1. Thus, the allocation (s_{j1}, \dots, s_{jn}) computed by the OPTIMAL algorithm is the optimal solution of OP_j . \square