Chapter 4
Static Load Balancing of a Computational Grid Using Competitive Equilibrium Approach

This chapter presents static approach for load balancing a computational grid using competitive equilibrium approach.

4.1. Introduction
Since computational resources are distributed and used by many users having different requirements, users are likely to behave in a selfish manner. Therefore, from the optimization perspective load balancing can be formulated as either cooperative load balancing ([18],[30],[83],[97]) or non-cooperative Nash game ([17],[38],[80]). While cooperative load balancing may not optimize individual response times simultaneously, Nash equilibrium solution may not achieve system optimal efficiency. In this study, we propose Competitive Equilibrium load balancing, a pricing mechanism for achieving both system optimal efficiency and individual optimality simultaneously.

At first, load balancing problem is translated to Fisher’s market model, where buyers are users and goods are computing resources. The Competitive equilibrium load balancing problem is then, finding equilibrium prices, and
allocation of user jobs to computing resources at these prices, such that each user maximizes her utility, subject to her budget constraints.

4.2. Grid System Model
We consider a computational grid system consisting of \( n \) nodes (computing resources) shared by \( m \) users. The nodes are heterogeneous, that is they have different processing power. Each user \( j \) generates jobs with an average rate of \( \phi_j \) (jobs per second) according to Poisson process and independent of other users. The total arrival rate of the system is \( \Phi = \sum_{j=1}^{m} \phi_j \). We assume that all the user jobs are of same size. Each computing resource is modeled according to [80], as an M/M/1 queuing system [73] (a single server queue model with Poisson arrivals and exponentially distributed service times). Each computing resource \( i \), executes jobs at a rate of \( \mu_i \).

For stability, the total job arrival rate must be less than the total processing rate of the system:

\[
\sum_{i=1}^{m} \phi_j < \sum_{i=1}^{n} \mu_i
\]  

(4.1)
Chapter 4: Static Load Balancing of a Computational Grid Using Competitive Equilibrium Approach

The fraction $x_{ji}$ of user $j$ jobs are sent to the computing resources $i$ ($i=1,...,n$) to be processed there such that

$$x_{ji} \geq 0,$$  \hspace{1cm} (4.2)

$$\sum_{i=1}^{n} x_{ji} = 1$$  \hspace{1cm} (4.3)

For stability, the load fractions of various jobs sent to computing resource $i$ must not exceed the rate at which jobs can be executed at processor $i$

$$\sum_{j=1}^{m} x_{ji} \phi_j \mu_i < \mu_i$$  \hspace{1cm} (4.4)

Each user $j$ ($j=1,...,m$) must find the load fractions $x_j = (x_{j1},...,x_{jn})$, such that each users response time is minimized independent of others subject to the constraints (4.2) to (4.4). We call $x_j$, the strategy profile of user $j$, and $x=(x_1,...,x_n)$ the global strategy profile.

For an M/M/1 queuing system [73], the expected response time at computing resource $i$ is given by:

$$F_i(x) = \frac{1}{\mu_i \lambda_i}$$  \hspace{1cm} (4.5)

where $\lambda_i$ is the average arrival rate of jobs (in jobs per second) at node $i$. That is, each computing resource receives
jobs from multiple users; therefore, $\lambda_i$ is a combination of job arrivals from various users. Thus expected response time at node $i$ is given by

$$F_i(x) = \frac{1}{\mu_i \cdot \sum_{j=1}^{m} x_{ji} \varphi_j}$$

(4.6)

The overall response time of user $j$ jobs is given by

$$D_j(x) = \sum_{i=1}^{n} x_{ji} F_i(x)$$

(4.7)

$$= \sum_{i=1}^{n} \frac{x_{ji}}{\mu_i \cdot \sum_{k=1}^{m} x_{ki} \varphi_k}$$

Clearly, $F_i(x)$ and $D_j(x)$ are strictly increasing, convex, and continuously differentiable functions of $x_j$.

The best response time for user $j$ job is a solution to the following optimization problem

$$\min_{x_j} D_j(x)$$

subject to the constraints (4.2) to (4.4)

The mean response time of all jobs is given by:

$$D(x) = \frac{1}{\varphi} \sum_{j=1}^{m} \varphi_j D_j(x)$$

(4.9)

which is equivalent to
4.3. Competitive Equilibrium Load Balancing

The competitive equilibrium load balancing problem consists of finding a set of prices and allocation of jobs to computing resources such that each user maximizes her utility, subject to her budget constraints, and the market clears (i.e., all money is spent).

We start by translating our grid system to Fisher’s market model, where buyers are users and goods are computing resources. Each user $j$ ($j = 1, \ldots, m$) is endowed a budget $w_i > 0$ to purchase computing power, where $w_i$ is not real money but artificial and can be interpreted as “importance weight”. When $w_i = 1$ for all users, then all users are treated uniformly important. Also, each user $j$ has a utility function $u_j(x) = -D_j(x)$ as given in equation (4.7), which represents her preferences for the different bundles of goods. The price of unit job executed on node $i$ is $p_i$; $p_i$ like $w_i$ is not real but artificial money and can be interpreted as “preference or ranking” of computing resource.
At given prices \( \mathbf{p} = (p_1, \ldots, p_n) \), each user \( j \) must find the load assigned to each node \( i (i = 1, \ldots, n) \), such that her utility function \( u_j(x) \) is maximized, under her budget constraints (i.e., \( \sum_{i=1}^{n} p_i x_{ji} \phi_j \leq w_j \)) and decisions by all other users.

Equilibrium is a vector of prices \( \mathbf{p} = (p_1^*, \ldots, p_n^*) \) at which for each user \( j \) there is a load assignment \( x_j^* = (x_{j1}^*, \ldots, x_{jn}^*) \) such that the market clears. That is for each user \( j \), the vector \( x_j^* \) is a maximizer of

\[
\max_{x_j} \quad u_j(x^*)
\]

subject to the constraints (4.2) to (4.4) and market clearing condition given by

\[
\sum_{i=1}^{n} p_i^* x_{ji}^* \phi_j \leq w_j
\]

Clearly, the utility function for each player \( u_j(x) \) is strictly continuous, concave, and continuously differentiable function of \( x_j (j = 1, \ldots, m) \), where \( x_j \subseteq \mathbb{R}_+^n \), and \( x_j \) is a closed convex set, bounded from below. The necessary and sufficient conditions for the existence of
Chapter 4: Static Load Balancing of a Computational Grid Using Competitive Equilibrium Approach

Competitive equilibrium based on the lemma of abstract economy developed by Debreu [25] are satisfied, hence there exists a Competitive equilibrium for the load balancing problem.

This is an artificial exchange of money where the price $p$ and budget $w$ are not real money and do not have any physical interpretations. The meaningful output is only load distribution $x$. The $w$ and $p$ have no outside use; they are only an economic means for defining user’s strategy profile to achieve individual and system optimality.

Walras [54] introduced a price-adjustment process called tâtonment trial and error process run by a fictitious auctioneer. The buyers take the prices as given, and report their demands at these prices to the auctioneer. The auctioneer, then adjusts prices in proportion to the magnitude of the aggregate demands, and announces the new prices. In each iteration the buyer recalculates their demands upon receiving the newly adjusted prices and reports these new demands to the auctioneer. The process continues until prices converge to equilibrium.
Chapter 4: Static Load Balancing of a Computational Grid Using Competitive Equilibrium Approach

We present below the algorithm for computing Competitive equilibrium solution (CES) for the load balancing.

4.3.1. Algorithm (CES)

Input

Node processing rates: \( p_1, \ldots, p_n \)

Job arrival rates: \( \varphi_1, \ldots, \varphi_m \)

Output

Load allocations to the nodes: \( x_{j1}, \ldots, x_{jm} \) \( \forall j = 1, \ldots, m \)

1. Initialization

1.1. \( w_j \leftarrow 1 \) for all \( i = 1, \ldots, n \)

1.2. \( p_i \leftarrow 1/n \) for all \( i = 1, \ldots, n \)

2. Loop

2.1. At prices \( p_1, \ldots, p_n \) compute \( x_1, \ldots, x_m \) such that each user maximizes her utility function (4.11) subject to the constraints (4.1) to (4.4)

2.2. Obtain market clearing error \( a \), given as

\[
a = \left( \sum_{j=1}^{m} \xi_j^2 \right)^{1/2}
\]

where \( \xi_j \) is given by

\[
\xi_j = w_j - \sum_{i=1}^{n} p_i x_{ji} \varphi_j
\]

2.3. Adjust the prices \( p_1, \ldots, p_n \) in proportion to aggregate demands until \( a \leq \) error tolerance

4.4. Experiments

We ran a computer model to analyze the effects of different schemes on mean response time of all jobs and individual
response time of each job. The proposed algorithm (CES) and two other load balancing schemes are implemented for comparison purposes.

i. **Global Optimal Scheme** (GOS): This scheme minimizes the mean response time of all jobs. The loads $x_{ji}$ for each user are obtained by solving the following optimization problem

$$
\min_{x} D(x) \quad (4.14)
$$

subject to the constraints (4.2) to (4.4).

ii. **Nash Game Scheme** (NGS): In this scheme each user $j \ (j = 1, \ldots, m)$ must find the load $x_{ji}$ assigned to each computing resource $i \ (i = 1, \ldots, n)$ such that the response time of his own jobs is minimized. The best response time of user $j$ is a solution to the optimization problem given by (4.8).

The algorithm for NGS is run by initializing strategy $x_i$ of each player $i$ to zero vector. Each player then updates its strategy $x_i$ in a sequential manner by solving the optimization problem (4.8). An interesting case occurs when no player can change his strategy $x_i^*$ and decrease its response
time by choosing a different strategy $\pi_i$ when the other user’s strategies are fixed. In this case, the system is said to reach Nash equilibrium.

We modeled a heterogeneous system consisting of 16 computers with service rates as shown in Table 4.1. The system has 16 users with job arrival fractions $q_j$ as given in Table 4.2.

**Table 4.1. Service rates of the computers**

<table>
<thead>
<tr>
<th>Computer</th>
<th>1-6</th>
<th>7-11</th>
<th>12-14</th>
<th>15-16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Rate (jobs/second)</td>
<td>10</td>
<td>20</td>
<td>50</td>
<td>100</td>
</tr>
</tbody>
</table>

**Table 4.2. Job arrival fractions of the users**

<table>
<thead>
<tr>
<th>User</th>
<th>1-2</th>
<th>3-5</th>
<th>6-10</th>
<th>11-14</th>
<th>15-16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_j$ (job arrival fractions)</td>
<td>0.2</td>
<td>0.1</td>
<td>0.04</td>
<td>0.02</td>
<td>0.01</td>
</tr>
</tbody>
</table>

### 4.4.1. Effect of System Utilization

In this set of experiments, we vary the system utilization ($\rho$) from 10% to 90%. The total job arrival rate of the system $\varphi$ is determined from the system utilization $\rho$ and the aggregate service rate of the system by fixing the system utilization in the following equation:
Chapter 4: Static Load Balancing of a Computational Grid Using Competitive Equilibrium Approach

\[ \rho = \frac{\varphi}{\sum_{i=1}^{n} \mu_i} \quad (4.15) \]

The job arrival rate for each user \( \varphi_j, j=1,\ldots,10 \) is determined from the total arrival rate as \( \varphi_j = q_j^* \varphi \).

Figure 4.1 shows the number of iterations required to reach the equilibrium at various system utilizations. In general the number of iterations required to reach equilibrium is not predictable.

![Figure 4.1 Convergence of CES Algorithm](image)

Figure 4.1 shows the number of iterations required to reach the equilibrium at various system utilizations. In general the number of iterations required to reach equilibrium is not predictable.

Figure 4.2 shows the mean response time of all jobs as the system utilization is varied from 10% to 90%. In all the three schemes, we see a rapid increase in the average response time of all users as the system nears full capacity.
Also, we see that the mean response time of all jobs is the least in GOS. Moreover, though the mean response time of all jobs in CES is greater than GOS, it is lesser than NGS. Thus, system optimal efficiency of CES is greater than NGS.

![Figure 4.2 Mean Response Time of all Jobs Vs System Utilization](image)

Figures 4.3, 4.4, and 4.5, show the response time of each player at system utilizations 10%, 60%, and 90% respectively. As can be seen from the graphs, in most of the cases, CES yields better performance than GOS and NGS. Thus, we observe that the individual optimality of CES is better than NGS.
4.4.2. Fairness of Schemes

Fairness index [77] is an important performance metric besides the mean response time. It is the measure of fairness of allocation of resources to users, and is given by

$$\text{FI} = \frac{\left( \sum_{i=1}^{m} D_j(x) \right)^2}{m \ast \sum_{i=1}^{m} (D_j(x))^2}$$  \hspace{1cm} (4.16)

$\text{FI}=1$, when the individual response time of each player is same. If the differences on $D_j(x)$ increases, FI decreases and the load balancing scheme favors only few.
Figure 4.4 Expected Response Time of Each Player at System Utilization of 60%

Figure 4.5 Expected Response Time of Each Player at System Utilization of 90%

Figure 4.6 shows the fairness index of the three schemes. It can be seen that fairness index of CES is greater than NGS and GOS and is close to 1.0. Fairness index of NGS is greater than GOS but lesser than CES.
4.5. Conclusions
Our study proposes competitive equilibrium solution for load balancing problem. It combines the system optimal efficiency of cooperative load balancing and individual optimality of non-cooperative load balancing. The mean response time in competitive equilibrium solution is close to system optimal solution and at the same time achieved better individual response times than non-cooperative Nash game solution.

An iterative and centralized algorithm for computing competitive equilibrium for load balancing is presented. Although this works well in our computational experiments, its convergence is unproven. A distributed parallel implementation of the algorithm can be tried in the future.