The Home Model and Competitive Algorithms for Load Balancing in a Computing Cluster

Ron Lavi* and Amnon Barak
The Hebrew University of Jerusalem, Israel†

Abstract

Most implementations of a Computing Cluster (CC) use greedy-based heuristics to perform load balancing. In some cases, this is in contrast to theoretical results about the performance of on-line load balancing algorithms. In this paper we define the *home model* in order to better reflect the architecture of a CC. In this new theoretical model, we assume a realistic cluster structure in which every job has a "home" machine which it prefers to be executed on, e.g. due to I/O considerations or because it was created there. We develop several on-line algorithms for load balancing in this model. We first provide a theoretical worst-case analysis, showing that our algorithms achieve better competitive ratios and perform less reassignments than algorithms for the unrelated machines model, which is the best existing theoretical model to describe such clusters. We then present an empirical average-case performance analysis by means of simulations. We show that the performance of our algorithms is consistently better than that of several existing load balancing methods, e.g. the greedy and the opportunity cost methods, especially in a dynamic and changing CC environment.

1 Introduction

A Computing Cluster (CC) is becoming a popular and powerful platform for various types of computation. A typical CC is composed of several heterogeneous machines connected by a high speed communication network, thus providing a cost-effective computation platform. One important factor for achieving good performance on such a platform is the load balancing method used. Most implementations of a CC use greedy-based heuristics for job assignments and reassignments. This is sometimes incompatible with the theoretical knowledge about the performance of on-line load balancing algorithms. This theoretical field has developed significantly in the last decade (the interested reader is referred to the survey [4]). Specifically, theoretical results [2, 3] indicate that when the cluster is not homogeneous, e.g. with respect

*Corresponding Author. E-mail: tron@cs.huji.ac.il.
†Authors' address: Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem 91904, Israel; email: {tron, amnon}@cs.huji.ac.il

to the machines' speed and memory, or with respect to the I/O demands of the jobs, then there are better algorithms than greedy.

One of the research attempts to bridge the gap between theory and practice is the opportunity cost approach that was developed using a theoretical basis [1, 7]. These works develop a framework for the management of several cluster resources as well as I/O and inter-process communication overheads. The advantage of the opportunity cost algorithms is realized mainly when jobs have complex characteristics, e.g. jobs that perform I/O to various machines, or that require both high CPU and memory usage. In [1, 7] it is shown, using theoretical analysis and by means of simulations, that the opportunity cost algorithms outperform greedy-based methods in such cases. However, when the environment is closely related, e.g. all jobs have small I/O overheads, a greedy approach still performs better. This is explained by the large gap between two basic theoretical models: the model in which machines are characterized only by their speeds, called *related machines* (i.e. all jobs have the same preference order of the machines), and the general model which does not assume any global characterization of the machines, called *unrelated machines* (i.e. each job may have a different preference order of the machines). Thus, since the character of jobs in an average CC varies between that of a "related" environment (where greedy performs better) and that of an "unrelated" environment (where opportunity cost performs better), it remains unclear which load balancing method to use.

In this paper, we make another step in this direction, trying to integrate between theory and practice. We characterize a new theoretical model – called the *home model* – to better describe the specific architecture of a CC. This model is intended to reflect a general CC architecture by introducing the assumption that each job is created on a specific machine, its *home machine*, and it prefers to be executed on that machine, e.g. because of I/O considerations. The home model is "located" between the above two models: it is more restrictive than the unrelated machines model and less restrictive than the related machines model. The home model is also motivated by several realistic cluster architectures. For example, in the MOSIX distributed operating system [5], processes communicate with the system and with other processes through their homes, and therefore prefer to be executed there. Another example is a computing cluster composed of several computing centers connected by a Wide Area Network. Since each process communicates mainly with its originating center it prefers to be executed there. In both examples, of-course, the need to migrate some processes to remote nodes exists, in order to balance the load and the usage of other resources.

We define the new model and develop two simple on-line load balancing algorithms for this model. We first provide a theoretical worst-case analysis of our algorithms. We show that they achieve better competitive ratios and perform less reassignments than the known algorithms for the unrelated machines model (and thus than the opportunity cost algorithms, which rely on them). We also show that the greedy algorithm does not perform well in the home model, from a theoretical point of view.

We then present an empirical average-case analysis of the performance of our algorithms, by means of simulations. We first examine several heuristics to improve their performance in a "real" average-case cluster. Next, we compare their performance to that of the greedy and the opportunity cost algorithms. We consider two kinds of dynamic cluster environments: in the first one, the dynamic nature is due to the changing I/O overheads, and in the second

one, the dynamic nature is due to the (changing) number of home machines. In both cases, the environment actually changes from a (nearly) related to a (nearly) unrelated environment. The simulation results show that our new algorithms perform consistently better than both greedy and opportunity cost, when the environment is either nearly related or nearly unrelated. For example, in many cases, the new algorithms are better than opportunity cost by about 10%-20%, and are better than greedy by as much as 40%.

The rest of this paper is organized as follows: In section 2 we describe and analyze theoretically the home model and the new algorithms. In section 3 we describe heuristic improvements to these algorithms, and examine their average-case performance with respect to existing load balancing methods. Our conclusions and suggestions for further research are given in section 4.

2 The Home Model

In this section we define the home model and describe two competitive algorithms for different variants of the problem. We also shown that the simple greedy approach does not perform well in this model.

2.1 Statement of the Problem

In the load-balancing problem, a sequence of n independent jobs arrive at arbitrary times, where every job must be assigned **immediately** to one of the m cluster machines. A job j is defined by its *load vector*, $(p_j(1), \ldots, p_j(m))$, where $p_j(i)$ is the weight of j on machine i (i.e. the load it creates on i). The *load*, l_i, of machine i is the sum of weights of all jobs currently assigned to i, i.e. $l_i = \sum_{k \in J_i} p_k(i)$, where J_i is the set of jobs assigned to i. Jobs may be either *permanent*, i.e. they never depart (or equivalently depart only after the last arrival), or *temporary*, i.e. they have a fixed (but unknown) duration. The goal of the load balancing algorithm is to minimize the maximal load seen during the running time.

Thus, in the load balancing problem, jobs are viewed as abstract objects with weights (possibly a different weight for each machine), and the algorithm receives a sequence of events: either a "job arrival" or a "job departure". This is different from scheduling problems, where it is usually assumed that jobs are characterized by the amount of work they need to execute. The "efforts" of the load balancing algorithm is therefore directed to balance the load of the cluster. This is especially useful since we do not assume any knowledge about the jobs character, besides the load they create. [1]

As usual, an on-line algorithm is evaluated by its *competitive ratio*, defined as follows: for some on-line load balancing algorithm A, let $L(\sigma)$ be its maximal load for the job sequence σ, and let $L^*(\sigma)$ be the maximal load of the optimal assignment of the jobs in σ. A is C-

[1] However, this implies that the duration of a job is independent of the current load. This is clearly not a realistic assumption. We use it only for theoretical analysis as it is the usual theoretical assumption. In our simulations we use the realistic assumption of fixed amount of work, i.e. the simulator creates the "job departure" event when the job completes some amount of CPU time, thus evaluating the performance under realistic conditions.

competitive if for **every** job sequence σ, $L(\sigma) \leq C \cdot L^*(\sigma)$. In other words, in the worst-case, A's maximal load is at most C times the lowest possible maximal load.

A machine model determines a specific form for the jobs' load vectors, e.g. in the identical machines model all the coordinates of a load vector are identical. We define a new machine model, called the home model, in which each job has a "home" machine which it prefers to be assigned to. More formally, the load vector of jobs in the home model is defined as follows:

$$p_j(i) = \begin{cases} p\_in_j & i = h_j \\ p\_out_j & \text{otherwise,} \end{cases}$$

such that $\forall j: \ p\_in_j < p\_out_j$. This property of higher remote load is observed in many real CCs [7], and is explained by the CPU processing overhead of the network protocols.

The only currently known algorithms for this problem are algorithms for the more general model of unrelated machines. These algorithms have logarithmic competitive ratios [2, 3, 4]. In this section we give algorithms with constant competitive ratios for two variants of the home model, unit and variable loads at home. The first variant is suitable when jobs are differentiated mainly by their outside load and create similar loads when assigned to their home. The second variant is suitable when jobs are differentiated by their local (home) load, and their outside load simply increases by some constant factor (due to the remote I/O effect). The techniques we use here are mainly based on those of Azar et al. [2] regarding load balancing for related machines.

In order to reduce the competitive ratio, we use a limited number of job reassignments, i.e. changing the assignment of a job from one machine to another. We note that experience with real systems shows that job reassignments may be performed with very small overhead, thus achieving a powerful tool for load balancing [5, 6]. Job reassignments are also used in the (theoretical) algorithm for temporary jobs on unrelated machines [3].

2.2 Unit Loads at Home

We consider a variant of the home model in which the load vector's form is:

$$\forall i, \ 1 \leq i \leq m \ : \ p_j(i) = \begin{cases} 1 & i = h_j \\ p_j & \text{otherwise} \ , \end{cases}$$

such that $\forall j: \ p_j > 1$. Machine h_j is the home machine of job j.

We first present an algorithm for permanent jobs, and then extend it to handle temporary jobs. We say that job j is heavier than job j' if $p_j > p_{j'}$ (similarly, j' is lighter than j). We denote the optimal off-line assignment by *Opt*, and the optimal maximal load by OPT.

Algorithm *AssignH* is based on an estimation of the optimal maximal load, OPT, and it indicates when this estimation is too low. We obtain a good estimation of OPT by using the doubling technique of [2], as follows: the *AssignH* algorithm is used as a procedure that receives the current estimation of OPT. We start with a very low (arbitrary) estimate, and double it each time the procedure returns "fail" (thus every job is eventually assigned). After such a failure we "reset" all machine loads, so that effectively the procedure ignores the load caused by jobs assigned before the last failure. Given this estimation, *AssignH* builds

a "good enough" job assignment by simply keeping all the heavy jobs in their homes, thus keeping the cluster not too overloaded.

Algorithm 1 (AssignH) *Receive Λ, an estimation of OPT, as a parameter. Upon arrival of job j perform the following steps:*

1. *If $l_{h_j} < 2\Lambda$, assign j to its home and return "success".*
2. *Otherwise, if there are non-local jobs assigned to h_j denote by j_out the heaviest one. If all jobs are local, denote by j_out the lightest one, including j.*
3. *If there is a machine i such that $l_i + p_{j\_out} \leq 2\Lambda$, then (re)assign j_out to i, assign j to its home (if $j\_out \neq j$), and return "success". Otherwise return "fail".*

Lemma 1 *If $\Lambda \geq OPT$, then, for the set of jobs that* AssignH *decides to assign outside their homes, there is an optimal assignment that assigns all these jobs outside their homes as well.*

Proof. Assume by contradiction that this is not true. Let j_out be the first job that *AssignH* decides to assign outside, and that all optimal assignments assign home. Denote the home machine of j_out as h. As a result of step 2 of *AssignH*, all jobs already assigned to h are local and $l_h > \Lambda \geq OPT$. Therefore there is some other job j_in that is assigned to h by *AssignH* and to some machine $i \neq h$ by *Opt*. Then swapping these jobs in *Opt* will not increase l_i since $p_{j\_in} \geq p_{j\_out}$, and will not change l_h since both j_in, j_out have unit loads in h. Thus we get an optimal assignment in which j_out is assigned outside. ∎

Lemma 2 *If $\Lambda \geq OPT$ then* AssignH *never returns "fail".*

Proof. Assume that *AssignH* failed when reassigning j_out. According to Lemma 1, *Opt* assigned j_out outside, thus $p_{j\_out} \leq OPT \leq \Lambda$. Since *AssignH* failed, $\forall i \ : \ l_i + p_{j\_out}(i) > 2\Lambda$ and thus $\forall i \ : \ l_i > \Lambda \geq OPT \geq l_i^*$, where l_i^* is the load of machine i in *Opt*. Denote by In, Out the set of jobs that *AssignH* assigned in/outside their homes, respectively, and by In^*, Out^* the appropriate sets of *Opt*. Therefore:

$$\sum_{j \in Out} p_j + \sum_{j \in In} 1 = \sum_i l_i > \sum_i l_i^* = \sum_{j \in Out^*} p_j + \sum_{j \in In^*} 1 \ ,$$

as both left and right equalities derive from the fact that the summation is over all jobs, but the summation order is different. Since $Out \cup In = Out^* \cup In^*$ and $\forall j : p_j > 1$ there must be some $j \in Out$ such that $j \notin Out^*$, which is a contradiction to Lemma 1. ∎

It can be verified that *AssignH* performs at most one reassignment whenever a new job arrives, thus performing a total of no more the n reassignments (an average of one reassignment per job). It is also the case that all the loads are always less than 2Λ. Thus if we start with $\Lambda = OPT$ then *AssignH* is 2-competitive. It is shown in [2] that the doubling technique for estimating OPT increases the competitive ratio by a factor of 4. We conclude:

Theorem 1 *Algorithm* AssignH *for permanent jobs is 8-competitive and performs at most n job reassignments.*

In order to handle temporary jobs we describe a method for job departures. As mentioned, we assume that job durations are fixed, so the sets of running jobs of the off-line and of (any) on-line algorithm are identical. Upon a departure of job j, if j was assigned to its home, the new departure method simply reassigns back home the heaviest job that is assigned outside and that its home is h_j.

Theorem 2 *Algorithm* AssignH *for temporary jobs is 8-competitive and performs at most* $2 \cdot n$ *job reassignments.*

Proof. We first claim that Lemma 1 still holds, if referring to the set of active jobs. We prove this by induction on the sequence of job arrivals and departures. Upon an arrival, the original proof holds. Upon a departure from machine h, after reassigning the heaviest job (whose home is h) back to h , it is the case that the heaviest local jobs are assigned to h. If there is some job *j_out* that is assigned outside by *AssignH* and to h by *Opt* then $l_h > OPT$, and there is some job *j_in* as in Lemma 1 such that swapping these jobs results in an optimal assignment according to the claim.

Lemma 2 still holds since it relies only on Lemma 1. Since job arrivals and departures cause at most one reassignment, the algorithm performs at most $2 \cdot n$ reassignments, and the theorem follows. ∎

2.2.1 An implementation remark

The *AssignH* algorithm is very easy to implement in a distributed operating system. Although it is presented as a centralized algorithm, the assignment decisions can be done in a distributed manner as follows: each job is created on its home machine and is first assigned there. If at some stage the home machine decides to reassign a job, it broadcasts a request to all the machines (specifying the job's load), and receives an "accept" answer from "non-loaded" machines, as detailed in the algorithm. The only necessary global parameter is Λ, the estimation of the optimal load, which may be broadcasted to all cluster machines upon an update (when a machine fails to reassign a locally assigned job). Such communication capacity is common even for greedy-based load balancing.

2.3 Variable Loads At Home

In this variant of the home model, the jobs' load vector is of the form:

$$p_j(i) = \begin{cases} p_j & i = h_j \\ x \cdot p_j & \text{otherwise} \end{cases},$$

for some (globally) fixed constant $x > 1$.

The insertion method for this case follows the same principle of minimizing the sum of weights of non-local jobs while keeping the machines not too overloaded. But since the home loads are now different, it is not possible to simply replace a job with a heavier one, as done before.

Algorithm 2 (AssignHv) *Upon an arrival of job j, first consider assigning j to its home. If the resulting load exceeds 2Λ, perform the following steps:*

1. *Reassign (one by one) non-local jobs to other machines until the sum of their weights is at least p_j, keeping all loads below 2Λ. Return "Fail" if some non-local job cannot be reassigned.*

2. *Assign j to h_j if (now) possible. Otherwise, denote by $p_1, \ldots, p_l$ all local jobs (in h_j) in non-decreasing order ($\forall i : p_i \leq p_{i+1}$). Let $k := min\{k \mid \sum_{i=1}^{k} p_i \geq p_j\}$. If $l_{h_j} - \sum_{i=1}^{k} p_i + p_j < \Lambda$ then decrease k by one.*

3. *If $p_k \geq p_j$ assign j outside, or "Fail" if this is not possible without exceeding 2Λ.*

4. *Otherwise, reassign jobs $1 \ldots k$ (one by one) to other machines and job j to its home, or return "Fail" if this is not possible for some job without exceeding 2Λ.*

Lemma 3 *Let Out, Out^* be the set of jobs assigned outside their homes by* AssignHv *and* Opt, *respectively. If $\Lambda \geq OPT$ then:*

$$\sum_{j \in Out} p_j \leq \sum_{j \in Out^*} p_j \ .$$

Proof. Suppose that an arrival of job j causes reassignment(s) of local job(s). Observe that all jobs assigned to h_j are local and that $l_{h_j} > OPT$. Let W be the weight of all local jobs already assigned outside by *AssignHv*. The weight of all local jobs exceeds OPT by at least $W + p_j$. Therefore, if *AssignHv* assigns j outside the lemma holds. If *AssignHv* reassigns the first $k-1$ lighter jobs outside, the lemma holds since $\sum_{i=1}^{k-1} p_i < p_j$. If job k is reassigned too, the weight of all local jobs exceeds OPT by at least $W + \sum_{i=1}^{k} p_i$, as verified in step 2. ∎

Lemma 4 *If job j is assigned outside by* AssignHv *and $\Lambda \geq OPT$ then $x \cdot p_j \leq OPT$.*

Proof. Examine the arrival of job j' that caused j to be assigned outside (possibly $j = j'$). It is the case that $p_{j'} \geq p_j$. Let In be the set of (local) jobs in h_j. Since $l_{h_j} + p_{j'} > 2 \cdot OPT$ and $p_{j'} \leq OPT$, then *Opt* must have assigned outside jobs with total weight at least $p_{j'}$ from $In \cup \{j'\}$. If *AssignHv* assigned j' outside, *Opt* must have assigned outside j' or a heavier job since there is no set of lighter jobs with total weight $\geq p_{j'}$ in $In \cup \{j'\}$, and the claim follows. Thus assume *AssignHv* assigned jobs $1 \ldots k$ (including j) outside. If *Opt* assigned j' (or a heavier job) outside then the claim follows. Otherwise *Opt* must have assigned outside all the jobs $1 \ldots k$, or some of them and a heavier job, and the claim follows. ∎

Lemma 5 *If $\Lambda \geq OPT$ then* AssignHv *never fails.*

Proof. Assume by contradiction that *AssignHv* fails to assign job j, which implies that $\forall i : l_i > OPT$ (using also Lemma 4). As in Lemma 2, we conclude:

$$x \cdot \sum_{j \in Out} p_j + \sum_{j \in In} p_j = \sum_i l_i > \sum_i l_i^* = x \cdot \sum_{j \in Out^*} p_j + \sum_{j \in In^*} p_j \ ,$$

and therefore $\sum_{j \in Out} p_j > \sum_{j \in Out^*} p_j$, which contradicts Lemma 3. ∎

In order to upper bound the number of reassignments made by *AssignHv*, we use the measure of *restart cost*, defined by Westbrook [8]. We assume that every job j has an associated restart cost $r_j = c \cdot p_j$ for a fixed constant c. Let $S = \sum_j r_j$, then every algorithm must incur a restart cost of at least S due to the initial assignment of all jobs. It is easy to verify that upon every arrival of job j, *AssignHv* reassigns jobs with total weight $< 2 \cdot p_j$. Thus, the total reassignment cost $< S + \sum_j 2 \cdot r_j = 3S$. By using the doubling technique to estimate OPT we conclude:

Theorem 3 AssignHv *for permanent jobs with variable loads is 8-competitive and incurs a total reassignment cost of at most* $3S$.

The method of handling temporary jobs is similar to that described for *AssignH*. Suppose job j has departed. *AssignHv* reassigns back home some jobs that are assigned outside, one by one, starting from the heaviest one, verifying that the home load does not exceed 2Λ, until the weight of the reassigned jobs is more than p_j (or until there are no candidate jobs). Observe that jobs heavier than j cannot be reassigned without overloading the home machine. Therefore, the total weight of jobs reassigned is at most $2 \cdot p_j$. We conclude:

Theorem 4 AssignHv *for temporary jobs with variable loads is 8-competitive. It incurs a total reassignment cost of at most* $5S$.

2.4 Related Machines in the Home Model

In order to refine the model to include a speed s_i to machine i (but still preferring the home machine), the general form of the load vector of job j is:

$$p_j(i) = \begin{cases} p\_in_j & i = h_j \\ p\_out_j / s_i & \text{otherwise} \end{cases},$$

such that $\forall i, j: \ p\_in_j < p\_out_j / s_i$.

We describe a general method to integrate the previous algorithms for the home model with the already known algorithms for related machines. We maintain two virtual sets of machines. The first set is associated to an algorithm H, which is the part of the home algorithm that includes only the decision mechanism whether to assign a job to its home or outside. The second set is associated to an algorithm for related machines, R, which assigns the jobs that H decided to move out of their homes. H and R operate independently (not considering the load created by the other). H may also retrieve a job from R in a case of job departure. Recall that *AssignH* never creates a load due to home jobs of more than $2 \cdot OPT$, that $\sum_{j \in Out} p_j < \sum_{j \in Out^*} p_j$, and that $\forall j \in Out$, either *Opt* assigned j outside or it assigned a heavier job outside instead. Thus, R can assign all jobs in Out creating a load of no more than $c \cdot OPT$, where c is the competitive ratio of R. Westbrook [8] describes an appropriate algorithm for temporary jobs with $c = 2 + \epsilon$ which incurs $O(S)$ reassignment cost. We get:

Theorem 5 *For related machines in the home model there is a* $(16+\epsilon)$*-competitive algorithm with* $O(S)$ *reassignment cost for both unit and variable loads.*

2.5 The Greedy algorithm

The intuitive greedy algorithm assigns a new arriving job j to machine i so that the resulting load, $l_i+p_j(i)$, is minimized. This algorithm is 2-competitive for identical machines, $\Theta(logn)$-competitive for related machines, and $\Theta(n)$-competitive for unrelated machines [2]. Thus, it is interesting to examine how "difficult" the home model is by examining the competitiveness of the greedy algorithm in this model. We use the method of [2] to show that their lower bound for unrelated machines is suitable for the home model with unit loads.

Lemma 6 *Greedy is not better than $(n-1)$-competitive in the home model with unit loads.*

Proof. For a group of $1 \ldots n$ machines, consider the following $0 \ldots n-1$ job arrivals:

$$h_0 = 1,\ p_0 = 1+\epsilon, \quad \forall j > 0:\ h_j = j,\ p_j = j + \frac{1}{j}\epsilon$$

Recall that the home loads of all jobs are 1. The greedy algorithm assigns job #0 to its home (machine #1), job #1 to machine #2 (since the resulting load in its home, machine #1, is 2 and the resulting load in machine #2 is $1+\epsilon < 2$), job #2 to machine #3 , and so on. The last job is assigned to machine #n for a resulting load of $(n-1) + \frac{1}{n-1}\epsilon$. The optimal assignment assigns each job to its home, except for job #0 which is assigned to machine #n, thus the optimal maximal load is $1+\epsilon$. Therefore the greedy to optimal maximal load ratio is no better than $n-1$. ∎

3 Performance Evaluation

We now present an empirical performance analysis of several algorithms in the home model, based on realistic scenarios, by means of simulations. The analysis is based on a simulator for a cluster of heterogenous machines connected by a LAN. The scenarios for the simulated executions are created randomly, by some distribution function, as follows: The machines' speeds are uniformly distributed between 1 and 2, the jobs' total required CPU time and arrival time are exponentially distributed with mean $2n$ and $8/n$, respectively, where n is the cluster size. Each job performs a fixed amount of work, a more realistic behaviour than the theoretical assumption of fixed duration. These are commonly assumed distributions in many simulation tests [7]. All the results are averaged over 100 simulated executions.

3.1 Heuristic Improvements to *AssignH*

We first consider several heuristics to improve the average-case performance of *AssignH*. One "weak point" of *AssignH* is the need to evaluate Λ, the optimal maximal load. Although a simple doubling scheme is sufficient for the theoretical proof, we consider several possible heuristics for this estimation method. First, we examine the effect of a more gradual increase of Λ instead of doubling it after a failure. We also examine the effect of considering all previously arrived jobs after Λ is increased (instead of ignoring them). When considering all jobs, it is also possible to (try to) fit back home some of the jobs that were assigned outside,

since Λ was increased. A possible drawback that should be regarded here is the effect of these heuristic improvements on the number of reassignments.

AssignH fails when all nodes are too overloaded, i.e. their loads exceed $2 \cdot \Lambda$. If the maximal $p_j(i)$ (over all i, j) was known (denote it by MAXP) then we could have considered a factor of $c = 1 + \text{MAXP}/\Lambda$ instead of 2 (actually adjusting the proof of Lemma 2). Since we do not have preliminary knowledge of MAXP, we examine a heuristic that estimates the value of MAXP in an adaptive manner.

Finally, we consider the effect of using a greedy method when deciding to reassign a job outside his home (instead of the method described in sub-section 2.4).

	Policy	Max. Load	Reassignments
1	Original (*AssignH*)	1.0	1.0
2	Orig. with Λ increase of $+1$	1.05	0.98
3	Considering previous jobs	1.06	1.58
4	Policy 3 with Λ increase of $*1.4$	0.94	2.3
5	P. 3 with Λ increase of $+1$	0.85	2.78
6	P. 5 with $c = 1 + \text{MAXP}/\Lambda$	0.84	3.0
7	P. 5 and moving back home after Λ increased	0.85	2.81
8	P. 5 and assigning outside jobs with greedy	0.85	2.34
9	A combination of policies 6,8	0.84	2.67

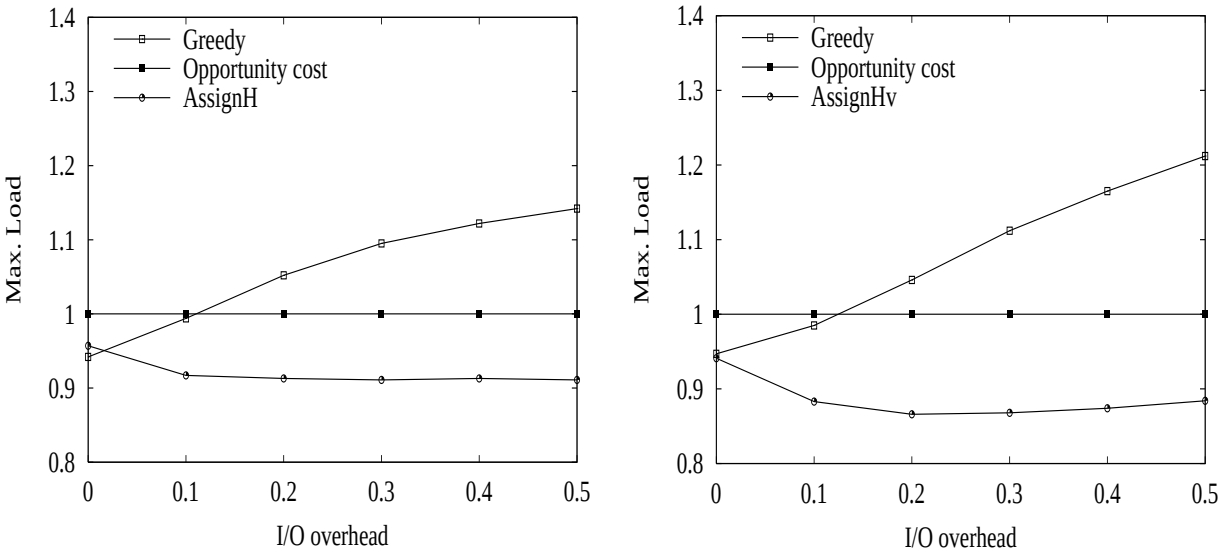
Table 1: The performance of several heuristics to improve *AssignH*

Table 1 compares the performance of these heuristics by simulations in the unit loads model, assuming a cluster of 16 machines. The results are normalized so that the first policy has a value of 1.0. As can be seen from the table, considering previously assigned jobs after Λ is increased combined with a Λ increase of $+1$ reduces the maximal load by an average of 15%. Refining c helps a bit more, and assigning outside jobs with greedy reduces the number of reassignments (and does not increase the maximal load). As also indicated, moving jobs back home after Λ is increased does not reduce the maximal load significantly. This may be some indication that *AssignH* chooses the "right" jobs to assign outside. Policy 9 summarizes the results, showing the performance of the heuristics that are implemented for the rest of this section.

We note that the original policy performed an average of one reassignment for every nine jobs. Results from real clusters, e.g. the MOSIX distributed operating system [5, 6], indicate that process migration is not very expensive, thus implying that such tradeoff of performance improvement and limited number of reassignments is worth while. We also note that all the load balancing algorithms in our simulations are allowed to make reassignments (e.g. greedy job migrations from loaded to less loaded machines) in order to examine the load balancing algorithm itself and not the effect of the job migrations.

3.2 Performance Measurements

This section compares the performance of three on-line algorithms: greedy, opportunity cost (o.c.) (as detailed in [7]), and *AssignH* (both for unit and variable loads). We first examine the effect of various levels of I/O overheads.



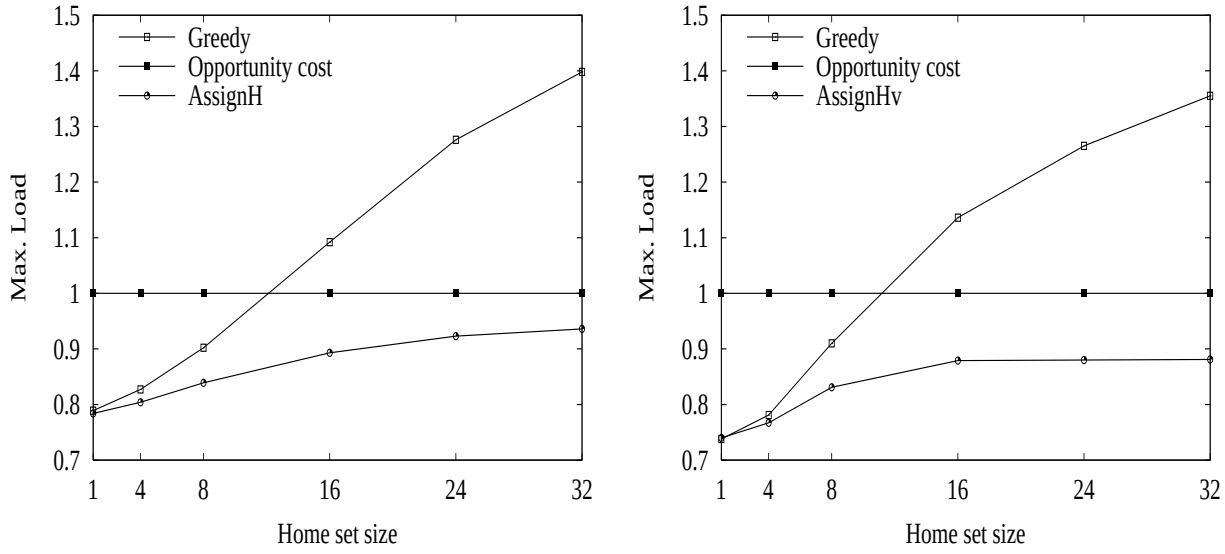
(a) Unit loads

(b) Variable loads

Figure 1: I/O overhead vs. Max. load

Figure 1 presents the maximal load of the three policies for different (average) I/O overheads. All the results are normalized according to the maximal load of the opportunity cost algorithm (i.e. its load is normalized to 1 and other loads are relative to it). For each graph point, the x-axis presents the mean value of an exponential distribution. If the I/O overhead of some job is x then the load of this job on a remote machine is $(1 + x) \cdot l$, where l is its home load. For variable loads, l is exponentially distributed with mean 1.0. We assume a cluster size of 32 machines.

From the figure it can be seen that the *AssignH* algorithms are consistently better than the o.c. algorithm in both models. For unit loads the maximal load of *AssignH* is lower by about 8%-10%, and for variable loads the load reduction is larger, by about 12%-14%. It can also be seen that the performance gap between *AssignH* and greedy increases along with the I/O overhead. For example, the maximal load of *AssignH* is lower by about 25% than that of greedy for large I/O overheads (of about 0.5) in the unit loads model. It can also be seen that *AssignHv* introduces a higher performance improvement than *AssignH*. It is interesting to observe the effect of the related vs. unrelated environments on the performance of the algorithms. While greedy is better than o.c. for a related environment (small I/O overheads) and vice versa for an unrelated environment, the new *AssignH* algorithms are consistently better than both greedy and o.c. for all I/O overheads.



(a) Unit loads

(b) Variable loads

Figure 2: Homes subset size vs. Max. load

In Figure 2 we examine the case where the set of home machines is a subset of the cluster, i.e. the home of a job is chosen from some subset of the machines, so only part of the cluster machines are home machines. The I/O overhead in this test case is exponentially distributed with a mean of 2.0. The cluster size is 32, while the size of the home machines subset ranges from 1 to 32. We note that as the homes subset is smaller, the machines are "more" related: when there is only one home (all jobs have the same home), the machines are exactly related. Thus, it is expected that greedy will perform better than o.c. for small homes subsets, and vice versa.

The conclusions from this test case are similar to those of the previous one, strengthening the implied conclusions from the performance evaluation. Specifically, as shown in figure 2, the *AssignH* algorithms perform consistently better than both the other two load balancing methods for all homes subset sizes. For example, for small home subsets, *AssignH* is better than o.c. by 20%, and for large home subsets, *AssignH* is better than greedy by 40%. It is also evident again that the advantage of the new algorithms is more significant in the variable loads model, by about 5%. It is interesting that the effect of the related or unrelated environment on the load balancing methods is more evident in this case: greedy performs significantly better than o.c. for small home subsets and vice versa for large homes subsets, while the new *AssignH* algorithms always perform better than both of these methods.

To summarize, the simulation results demonstrate two advantages of the *AssignH* algorithms. First, the new algorithms reduce the maximal load, both compared with the o.c. algorithm (by about 10%-15%), and compared with the greedy algorithm (by about 40%). Second, they have a consistent behaviour in a dynamic and changing environment, while the other two algorithms fit well only to a specific environment. This advantage exists both

when the dynamic nature is due to the set of home machines and when it is due to various I/O overheads.

4 Conclusions

This paper presents the home model for load balancing in a Computing Cluster (CC). It is a variant of the load balancing theoretical framework. This model fits more accurately to a typical cluster architecture, where the nodes are connected by a LAN, and when the main resources are CPU and I/O. We present two new on-line load balancing algorithms for this model and analyze their competitive ratios. We show that, from a theoretical point of view, they are better than both the the opportunity cost approach and the greedy method, to perform load balancing in the home model.

We also present a performance evaluation, by means of simulations, comparing the new algorithms to the greedy and the opportunity cost algorithms. The results show that the new algorithms are consistently better than the previous methods. Their behaviour is consistent in a dynamic and changing environment, while the other two algorithms fit well only to a specific environment. This advantage exists both when the dynamic nature is due to the set of home machines and when it is due to various I/O overheads.

Further research about the home model may be performed in several directions. A more theoretical understanding of the home model is desired: are there better load balancing algorithms for this model, or can we find some lower bounds to prove otherwise? Another interesting theoretical direction may be to expand the definition of the model, aiming to reflect more complex cluster structures. For example, a model in which a job has several homes with different preferences. Regarding all the machines in the cluster as homes with different preferences is actually very close to the unrelated machines model. Finally, it is interesting to extend the model to the direction of a Wide Area Cluster Computing. This setting is different since outside loads are lower than local loads because of network latency, although it seems natural to use similar algorithms to the ones described in this paper.

Acknowledgments

We wish to thank Yossi Azar and Arie Keren for helpful discussions, and Ahuva Mu'alem and Iris Shochet for helpful comments on an early draft of the paper.

References

[1] Y. Amir, B. Awerbuch, A. Barak, R.S. Borgstrom, and A. Keren. An opportunity cost approach for job assignment and reassignment in a scalable computing cluster. In *Proc. of the 1998 International Conference on Parallel and Distributed Computing and Systems (PDCS'98)*, pages 639–645, October 1998.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, 1997.

[3] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts. Competitive routing of virtual circuits with unknown duration. In *Proc. of the 5th ACM-SIAM Symposium on Discrete Algorithms (SODA'94)*, pages 321–327, 1994.

[4] Y. Azar. On-line load balancing. In A. Fiat and G. Woeginger, editors, *Online Algorithms: The State of Art*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

[5] A. Barak and O. La'adan. The MOSIX multicomputer operating system for high performance cluster computing. *Journal of Future Generation Computer Systems*, 13(4–5):361–372, 1998. http://www.mosix.cs.huji.ac.il.

[6] A. Barak, O. La'adan, and A. Shiloh. Scalable cluster computing with mosix for linux. In *Proc. of the 5th Annual Linux Expo*, pages 95–100, May 1999. http://www.mosix.cs.huji.ac.il.

[7] A. Keren. *On-line Assignment of Processes in a Scalable Computing Cluster*. PhD thesis, Institute of Computer Science, The Hebrew University of Jeruslaem, Israel, 1998.

[8] J. Westbrook. Load balancing for response time. In *Proc. of the 3rd Annual European Symposium on Algorithms (ESA'95)*, 1995.