Competitive Home Model Algorithms for Load Balancing in a Computing Cluster

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

Abstract

Most implementations of a Computing Cluster (CC) use greedy-based heuristics to perform load balancing. This is in contrast to the theoretical knowledge 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 a theoretical model. We develop several on-line algorithms for load balancing in this model. Our algorithms achieve better competitive ratios and perform less reassignments than the algorithms for the unrelated machines model, which is the best model known so far to describe clusters with complex characteristics, e.g. due to complex I/O patterns. We also present empirical analysis of the performance of the new algorithms, showing that they perform consistently better than existing load balancing methods, especially in a dynamic and changing 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. In order to achieve good performance on such a platform there is a need to balance the load among the cluster machines. Most implementations of a CC use greedy-based heuristics for job assignments and reassignments. This is in contrast to the theoretical knowledge about the performance of on-line load balancing algorithms, which has developed significantly in the last decade. Results from this field indicate that when the cluster is not homogeneous, e.g. with respect to the machines' speed and the I/O demands of the jobs, then there are better algorithms than greedy.

Despite of these results, there has not been many research in order to bridge the gap between theory and practice. One exception is the opportunity cost algorithms that was developed using a theoretical basis [1, 7]. These works developed a theoretical framework for the management of several cluster resources as well as I/O and inter process communication overheads, based on algorithms for the model of unrelated machines. They have also shown, by means of simulations, that their algorithms outperform greedy-based heuristics in many common cases.

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

In this paper, we develop a theoretical model to better reflect the architecture of a CC - the home model. This new model is intended to better 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. This model is 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 execute their. In this case, if the cluster machines are connected by a LAN, the load the process creates in its home is lower than the load it creates on other machines, since the I/O overhead is due to the CPU processing of the network protocols [7]. 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 execute their. In both example, ofcourse, the need to migrate some processes to remote nodes exists, in order to balance the load and the usage of other resources (e.g. memory).

Introducing the home model is needed due to a large gap between two basic theoretical models: the model in which machines are characterized only by their speeds, called *related machines*, and the most general model which assumes no known characterizations of the machines (but achieving relatively weak worst case bounds), called *unrelated machines*. The home model is located between these two models, i.e. it is more restrictive than the unrelated machines model and less restrictive than the related machines model.

We define the model and develop several theoretical on-line algorithms for load balancing in this model. Our algorithms achieve better competitive ratios and perform less reassignments than the algorithms for the unrelated machines model and the opportunity cost algorithms, which rely on the algorithms for the unrelated machines model. We also show that the greedy algorithm performs poorly in the home model, from the theoretical point of view.

After obtaining theoretical algorithms for the home model, we perform empirical analysis of their performance by means of simulations. Their performance is compared to that of the greedy and the opportunity cost algorithms, showing that the new algorithms perform consistently better. We show that the main advantage of our algorithms is their improvement of the the main "weak point" of the opportunity cost algorithms. The advantage of the opportunity cost algorithms is realized mainly for jobs with complex characteristics, e.g. jobs that perform I/O to various machines. In this case it is not possible to assume that the fastest machine in the cluster is the preferred machine for all jobs, since this depends on their I/O requirements. However, when the environment is closely related, e.g. all the jobs have small I/O overheads, a greedy approach still performs better than the opportunity cost approach. The algorithms we develop do not suffer from this drawback. We show that they are better than the greedy method for related environments and better than the opportunity cost method for unrelated environments.

2 The Home Model

This section defines the home model and describes two competitive algorithms for different variants of the problem. It is also shown that the popular greedy method performs poorly in this model.

2.1 Statement of the Problem

The general load-balancing problem is defined as follows. We are given n machines and a sequence of independent jobs that arrive at arbitrary times. A job j is defined by its *load vector*, $p_j = (p_j(1), p_j(2), \cdots, p_j(n))$, where $p_j(i)$ is the *load* (l_i) of job j on machine i. When a job arrives it is assigned immediately to one of the machines, in an on-line manner, thereby increasing the *load* of this machine by the corresponding value of its load vector. The goal of the algorithm is to minimize the maximal load seen in the entire running time of the machines.

A machine model determines a specific form for the load vector, e.g. in the identical machines model all coordinates of the 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 $\forall j: \ p\_in_j < p\_out_j$.

The only currently known algorithms for this problem are algorithms for the more general model of unrelated machines. These algorithms have logarithmic competitive ratio [4]. In the following sections we give on-line algorithms for two variants of the home model, unit and variable loads at home, with constant competitive ratios. For this purpose, we extend techniques 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. moving a job from one machine to another after it has started to execute. We note that experience with real systems shows that job reassignments may be performed with very little overhead, thus achieving a powerful tool for load balancing [5, 6]. Job reassignments are also used in the theoretical model of 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 (i.e. jobs that never finish), and then extend it to handle temporary jobs. A job j is heavier than job j' if $p_j > p_{j'}$ (similarly, j' is lighter than j). The optimal off-line assignment is referred to as Opt and the optimal maximal load is denoted by OPT.

Algorithm 1 (assignH) *Receive Λ, the estimation of OPT, as a parameter. Upon an 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 exists a machine i such that $l_i + p_{j\_out} < 2\Lambda$ then (re)assign j_out to i. If $j\_out \neq j$ assign j to its home. If there is no such machine return "fail", otherwise return "success".*

Lemma 1 *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, assuming that $\Lambda \geq OPT$.*

Proof. Assume by contradiction that j_out is the first job that assignH decides to assign to machine $i \neq h_{j\_out}$ and that all the optimal assignments assigned j_out to h_{j_out}. As a result of step 2, all the jobs assigned to h_{j_out} are local and $l_{h_{j\_out}} > OPT$. Therefore there is some j_in that is assigned home by assignH and outside by Opt. Since $p_{j\_in} \geq p_{j\_out}$ than swapping these jobs will result in an optimal assignment in which j_out assigned home. ■

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 $\forall i \ : \ l_i > \Lambda \geq OPT \geq l_i^*$. 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 is easy to verify that assignH never creates load that exceeds 2Λ, and that assignH performs at most one reassignment whenever a new job arrives, thus performing no more the n reassignments during the entire running time (i.e. in average, each job is reassigned once).

In order to estimate the optimal load, we use the doubling technique described in [2]. Briefly, we start with a very low estimate of OPT, which is doubled each time the algorithm fails. After such a failure the algorithm ignores the load caused by jobs assigned before the failure. This technique increases the competitive ratio by a factor of 4 [2]. Thus we conclude:

Theorem 1 *Algorithm assignH is 8 competitive with respect to load and performs at most n reassignments in the entire running time.*

In order to handle temporary jobs with unknown durations, we include a method for job departures. As usual in the load balancing framework, we assume that job durations are fixed, e.g. independent of the machines' loads (thus the set 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 method 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 with respect to load and performs at most 2n reassignments.*

Proof. We prove that Lemma 1 holds if referring to the set of active jobs by induction on the sequence of job arrivals and departures. Upon an arrival the original proof holds. Upon a departure, after reassigning the heaviest job back home there exists that the home is overloaded and the heaviest jobs are assigned home, thus it is possible to show contradiction similar to the way used in Lemma 1. Lemma 2 holds since it relies only on Lemma 1. Since job arrivals and departures cause at most one reassignment, the algorithm performs at most 2n job reassignments. ■

2.3 Variable Loads At Home

We examine a different version of the "home" model, which differs from the previous one by the jobs' load vector form, defined by:

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

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. Since the home loads are different, it is not possible to simply replace a job with a heavier one. The method is, therefore, as follows:

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, in any order) non-local jobs until the sum of their weights $\geq p_j$ (or all of them) to other machines, maintaining the loads below 2Λ. "Fail" if this is not possible. Assign j to h_j if (now) possible.*
2. *Otherwise, if j is the lightest between all local job assign j to another machine as before, returning "fail" or "success" accordingly.*
3. *Otherwise, assume that the local jobs are ordered, i.e. $\forall i: \ p_i \leq p_{i+1}$ and let $k \leftarrow min\{l | p_l < p_j, \ \sum_{i=1}^{l} p_i \geq p_j\}$. If $l_{h_j} - \sum_{i=1}^{k} p_i + p_j < \Lambda$ then decrease k by one. 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 without exceeding 2Λ.*

Lemma 3 *Let Out, Out^* be the set of jobs assigned outside their homes by assignHv and Opt, respectively. Then it exists that:*

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

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 3. ∎

Lemma 4 *If job j is assigned outside its home by assignHv then $x \cdot p_j \leq OPT$.*

Proof. Examine the arrival of job j' that caused j to be assigned outside (possibly $j = j'$). Let In be the set of local jobs in h_j. Since $l_{h_j} > OPT$, Opt must assign jobs with weight at least $p_{j'}$ outside from the set of jobs $In \cup \{j'\}$. If assignHv assigned j' outside, Opt must assign outside j' or a heavier job since there is no set of lighter jobs with weight $\geq p_{j'}$. Assume assignHv assigned jobs $1 \ldots k$ outside. If Opt assigned j' home then it must have assigned outside all the jobs $1 \ldots k$, or some of them and a heavier job, which implies the lemma. Otherwise Opt assigned j' outside and $\forall \ 1 \leq i \leq k, \ p_i < p_j$. ∎

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 give an upper bound for the amount of reassignments made by assignHv, we use the measure of **restart cost**, defined by Westbrook [8]. Briefly, it is assumed there 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 with respect to load and incurs a total reassignment cost of at most $3S$.*

The method of handling temporary jobs is similar to that described for assignH. Suppose that 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 at least p_j (or until there are no candidate jobs). Observe that jobs that are heavier than job j can not 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(v) 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 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 $(12+\epsilon)$-competitive algorithm with $O(S)$ reassignment cost for unit loads, and a $(16+\epsilon)$-competitive algorithm with $O(S)$ reassignment cost for 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]. We use the method of [2] to show that their lower bound for unrelated machines is suitable for the home model with unit loads. It is interesting to examine how "difficult" the home model is by examining the competitiveness of the greedy algorithm in this model.

Lemma 6 *Greedy is $\Omega(n)$-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 $(n-1) + \frac{1}{n-1}\epsilon = O(n)$. The optimal assignment assigns each job to its home except for job #0 which is assigned to machine #n, thus the optimal load is $1 + \epsilon$. ∎

3 Performance Evaluation

This section presents an empirical performance analysis of several algorithms in the home model, based on realistic scenarios, by means of simulations.

The scenario for each simulated execution is created randomly, by some distribution function. 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 fixed amount of work, which is more realistic than the theoretical assumption of fixed duration. The results are averaged over 100 executions, and are normalized with respect to one of the examined algorithms.

3.1 Heuristic Improvements to *AssignH*

We consider several heuristics to improve average case performance. One "weak point" of AssignH is the need to evaluate Λ. A possible heuristic is to perform a more gradual increase of Λ instead

of doubling it after a failure. Another heuristic is to consider all previously arrived jobs after an increase. When considering all jobs, a third heuristic is to fit back home some of the jobs that were assigned outside, since Λ was increased. A possible drawback that should be regarded is the effect of these heuristics on the number of reassignments.

AssignH fails when all nodes are too overloaded, i.e. their loads exceed $c \cdot \Lambda$, where $c = 2$. We note that if the maximal $p_j(i)$ (over all i, j) is known (denote it by MAXP) than we could set $c = 1 + \text{MAXP}/\Lambda$ by adjusting Lemma 2. Since we can not assume preliminary knowledge of MAXP, we examine a heuristic that estimates the value of MAXP in an adaptive manner.

We also consider the effect of assigning jobs outside their homes in a greedy manner, as opposed to the method described in 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 5,6,8	0.84	2.67

Table 1: The performance of several heuristics

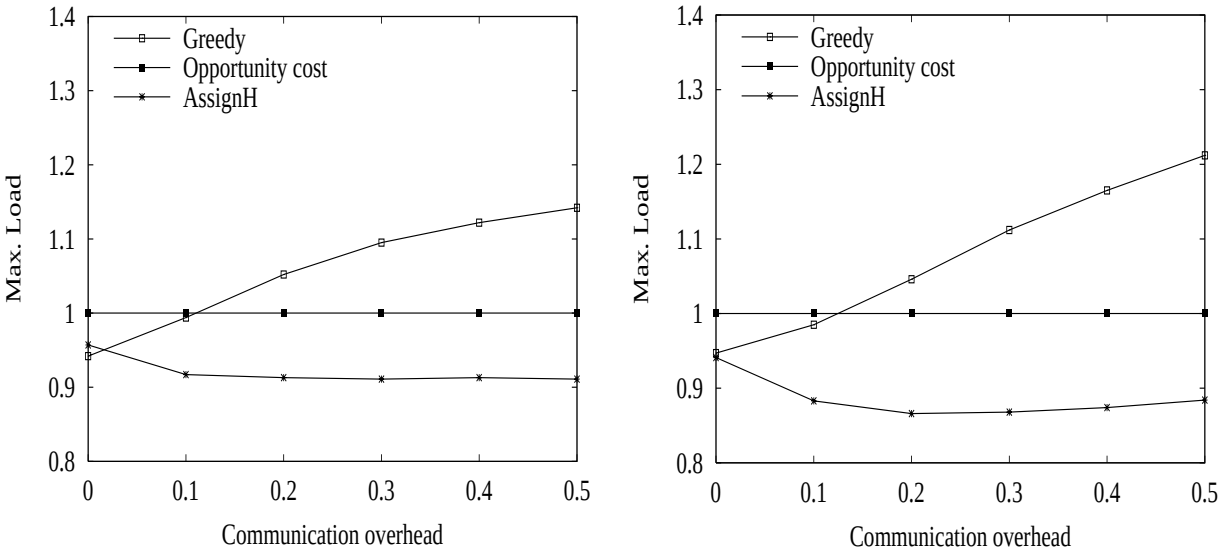
Table 1 compares the performance of these heuristics by simulations in the unit loads model, assuming a cluster of 16 machines. As can be seen from the table, considering previously assigned jobs after Λ is increased combined with a Λ increase of +1 reduces the maximal load by an average of 15%. Refining c helps a bit more, and assigning outside jobs with greedy reduces the number of reassignments (and does not increase the maximal load). We note that the original policy performed an average of one reassignment for every nine jobs. Results from real clusters [5, 6] show that process migration is not expensive, thus indicating that the tradeoff of performance improvement and more reassignments is worth while. Policy 9 shows the performance of the heuristics that are implemented in the rest of this section.

3.2 Performance Measurements

This section compares the performance of three on-line algorithms: greedy, opportunity cost (o.c.) and assignH(v).

Figure 1 presents the maximal load of the three policies for different (average) communication overheads. For each graph point, the x-axis presents the mean value of an exponential distribution. If the communication overhead of some job is x then the load of the job on a "non-home" machine is $(1 + x) \cdot l$, where l is its home load. For variable loads, l is exponentially distributed with mean 1.0. The cluster size is 32.

From the figure it can be seen that assignH(v) are consistently better than the o.c. algorithm



(a) Unit loads

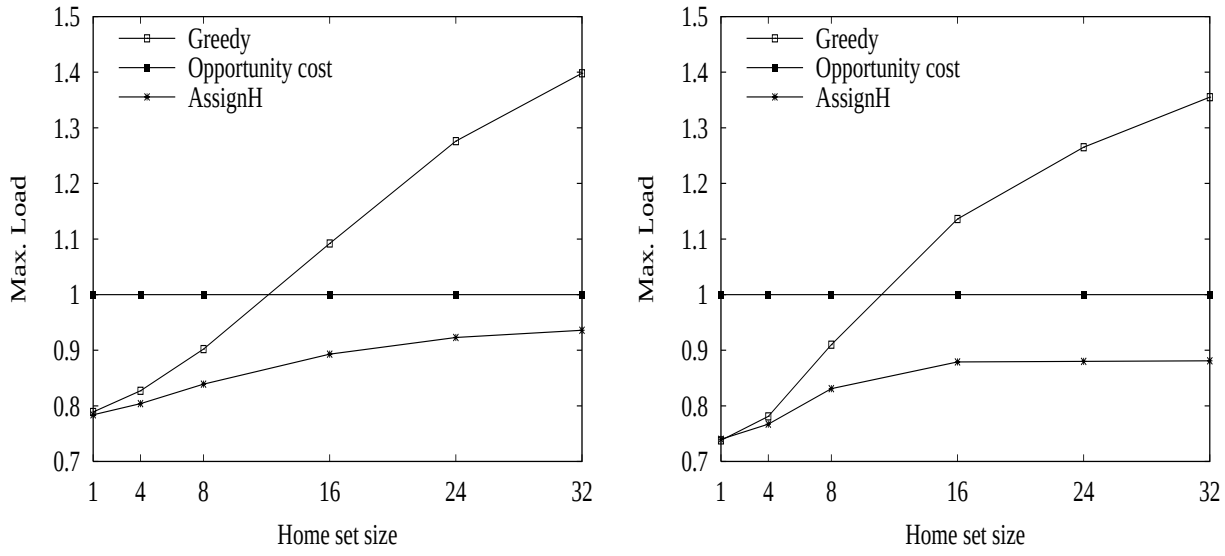
(b) Variable loads

Figure 1: Communication overhead vs. Max. load

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(v) and greedy increases along with the communication overhead. For example, the maximal load of assignH is lower by about 25% than that of greedy for a communication overhead of 0.5 for unit loads. In general, it can be seen that assignHv introduces a higher performance improvement. We note that assignH(v) are consistently better than greedy, even for small communication overheads. This is in contrast to the o.c. algorithm, which performs better than greedy only for large communication overheads.

Next we examine the case where the set of home machines is a subset of the cluster, i.e. only a subset of the cluster machines are homes for some jobs. The communication overhead here is exponentially distributed with mean 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, and when there is only one possible 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. Figure 2 presents the simulation results for this case. The results show that assignH(v) are consistently better than the other two policies for all homes subset sizes. More specifically, 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 that the advantage of our algorithms is more significant in the variable loads model, by about 5%.

The simulation results demonstrates two advantages of assignH(v). First, they reduce the maximal load, compared with the o.c. algorithm by about 10%-15%. They also reduce the maximal load of the greedy algorithm by as much as 40%. Second, they have a consistent behaviour in a



(a) Unit loads

(b) Variable loads

Figure 2: Homes subset size vs. Max. load

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 communication overheads. While greedy is better than o.c. for small communication overheads and/or small home set sizes, and vice versa for respectively large values, assignH(v) are better from both these algorithms for all such cases.

4 Conclusions

This paper presents the home model for load balancing in a Computing Cluster (CC). This model 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 on-line algorithms with constant competitive ratios, thus improving the $O(logn)$ competitive ratio of the opportunity cost approach. Our algorithms also perform less reassignments than the opportunity cost approach (for temporary jobs). It is also shown that the greedy method performs poorly in this method. 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 previous approaches. 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 communication overheads.

Further research about the home model may be performed in several directions. A more theoretical understanding of the home model is desired, especially regarding some lower bounds of

the competitiveness of algorithms for this model. 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 here.

Acknowledgments

We wish to thank Yossi Azar and Arie Keren for helpful discussions. Ron Lavi also thanks Noam Nisan for his moral encouragement.

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. 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. 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. 5-th 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 *3rd Annual European Symposium on Algorithms (ESA'95)*, 1995.