On-line Load Balancing *

Matthew Andrews† Lisa Zhang‡

In the on-line load balancing problem, jobs arrive in and depart from a system of m machines on-line and the scheduler has to assign each incoming job to one of the machines. (See Figure 1.) The load on the machine selected is then increased by a certain amount until the job leaves the system. The general goal is to minimize the maximum load. Load balancing problems arise naturally in various settings in which jobs have to compete for the available resources. Machines may represent various kinds of communication channels with certain bandwidth, and jobs requests for bandwidth. Alternatively, machines may represent distributed database platforms, and the jobs may be application programs accessing the database; here, the load represents the time for an access.

The problem we consider can formally be defined as follows. A set of jobs arrive in and depart from a system of m machines on-line. Each job j has a *weight*, w_j, which measures the level of service needed for the job, and a *reassignment cost*, r_j, which measures the cost of assigning (or reassigning) the job to a machine. Each machine i has a capacity, cap_i, which measures the ability of a machine to serve jobs. When job j arrives in the system, the on-line scheduler is informed of its weight and reassignment cost (but not of its departure time) and has to assign job j to some machine i at a cost of r_j. At any time, the scheduler is allowed to reassign some of the jobs currently in the system to other machines. The corresponding reassignment cost is paid for each reassigned job. At any time t, the *active* jobs are those that have arrived but not yet departed. The load λ_i on machine i is defined as the sum of the weights of the active jobs on machine i divided by the capacity cap_i. The maximum load λ is the maximum of λ_i over all i. We aim to minimize the maximum load.

The performance of an on-line algorithm can be measured using competitive analysis. In the context of load balancing, most analyses that have been performed compared the maximum load at time t of the on-line algorithm to the maximum load at any time between 0 and t, the *peak load*, of the best off-line algorithm presented with the same sequence of job arrivals and departures. Westbrook [1], however, pointed out that a competitive analysis against peak load is somewhat unrealistic. He instead proposed the notion of competitiveness against *current load*. We say that an on-line algorithm is *α-competitive against current load* if, at all times, the maximum load λ is at most α times the lowest possible load for the active jobs. In general, competitiveness against current load is a much stronger notion than competitiveness against peak load. Even though the arrival or departure of a job may greatly affect the optimum assignment of jobs to machines, a competitive algorithm against current load needs to ensure that its maximum load before and after the arrival (or departure) is still within a factor of α of the corresponding optimum.

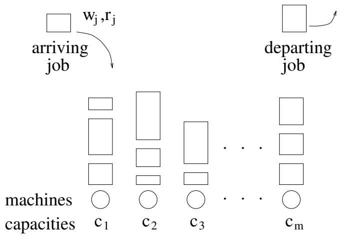

Figure 1: Jobs arrive in and depart from a system of machines.

If we don't allow reassignments of jobs then all algorithms have a poor competitive ratio against current load. On the other hand, if we allow an arbitrary amount of reassignment then the problem loses its on-line aspect. There is thus a trade-off between the competitive ratio and the amount of reassignment performed. We say that an on-line algorithm has a *reassignment factor of r* if, for all times, the total reassignment cost (including the initial assignments) is at most r times the sum S of the reassignment costs of all jobs that have arrived in the system (regardless of whether or not they have departed). A reassignment factor of 1 means that no reassignment is performed.

Results The bounds for our algorithms are presented in the following table. The left-hand column describes the various models considered. An entry (α, r) corresponds to an algorithm with competitive ratio α and reassignment cost r. If $\text{cap}_i = 1$ for all i then we say that the machines are *identical*. For identical machines we analyzed our algorithms for the cases of unit reassignment costs, reassignment costs that are

*This research is joint work with Professor Michel Goemans. For more details see http://theory.lcs.mit.edu/ ˜andrews

†MIT Laboratory for Computer Science and Department of Mathematics. Supported by NSF contract 9302476-CCR, ARMY grant DAAH04-95-1-0607 and ARPA contract N00014-95-1-1246. Advisor: Professor Michel Goemans. Email: andrews@theory.lcs.mit.edu.

‡MIT Laboratory for Computer Science and Department of Mathematics. Supported by an NSF graduate fellowship, ARMY grant DAAH04-95-1-0607 and ARPA contract N00014-95-1-1246. Advisor: Professor Tom Leighton. Email: ylz@theory.lcs.mit.edu.

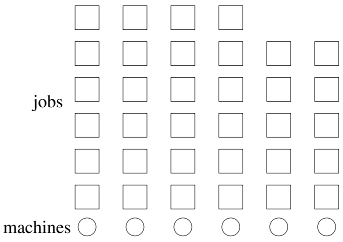

Figure 2: The algorithm used when job weights, reassignment costs and capacities are all 1.

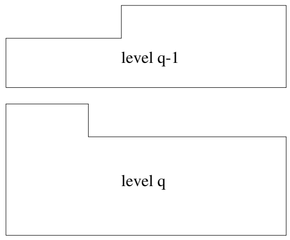

Figure 3: The level structure used for arbitrary job weights.

proportional to the job weights, and arbitrary reassignment costs. If the capacities are arbitrary then we say that the machines are *related*.

Model	Our results	Previous Results
$r_j = 1$	$(3.5981, 2)$	$(6, 2)$
$\text{cap}_i = 1$	$(3+\epsilon, f(\epsilon))$	
$r_j = bw_j$	$(3.5981, 3)$	$(6, 3)$
$\text{cap}_i = 1$	$(2+\epsilon, g(\epsilon))$	
r_j arbitrary $\text{cap}_i = 1$	$(3.5981, 6.8285)$	–
r_j arbitrary	$(32, 79.5)$	–
cap_i arbitrary	$(8+\sigma, 6.8285)$	

Two of the algorithms have bounds that are parameterized by ϵ. The reassignment factors are given by functions f and g that are increasing functions of ϵ. Hence these entries correspond the algorithms for which there is a trade-off between competitive ratio and reassignment factor.

The previous results are due to Westbrook[1]. He also considered the case where the machines are related and the reassignment costs are proportional to the weights. His algorithm has competitive ratio 24 and reassignment factor 4. Our algorithms are the first that are able to deal with arbitrary reassignment costs.

For the case of related machines and arbitrary reassignment costs we have two algorithms. The first has constant (although somewhat large) bounds. The second has competitive ratio $8+\sigma$ where $\sigma = 2\lceil\log(c_{\max}/c_{\min})\rceil$. Here, $c_{\max}$ is the maximum machine capacity and $c_{\min}$ is the minimum machine capacity. This latter algorithm is superior unless $c_{\max}/c_{\min} > 2^{12}$.

The Algorithms We now present the key ideas for some of the algorithms. First consider the simple case in which all machines have capacity 1, all job weights are 1 and all reassignment costs are 1. It is easy to maintain a situation in which there exists p such that all machines have either p or $p+1$ jobs. (See Figure 2). The maximum load is clearly optimal and the structure can be maintained in such a way that the reassignment factor is 2. To deal with arbitrary job weights we divide the jobs into levels. Level q contains jobs whose weights satisfy $\sqrt{3}^q \le w_j < \sqrt{3}^{q+1}$. Within each level the jobs are maintained in a structure similar to that shown in Figure 2. A key additional element is that the structures for consecutive levels are "symmetrically opposed". (See Figure 3.) This prevents the majority of the load from being assigned to the left-hand machines.

If the reassignment costs are arbitrary then maintaining the structures of Figure 2 is a little trickier. Within each level we divide the jobs into blocks based on their reassignment cost. Block b contains jobs whose reassignment costs satisfy $(1+\sqrt{2})^b \le r_j < (1+\sqrt{2})^{b+1}$. Whenever a block b job arrives or departs we reassign at most one job from each block $b' \le b$ and we do no reassignment of jobs in blocks greater than b. This enables us to bound the amount of reassignment performed.

The $8+\sigma$-competitive algorithm for related machines is similar in spirit to the above algorithms for identical machines. The 32-competitive algorithm uses different ideas and is based on the algorithm of Westbrook[1]. We again maintain a hierarchy of levels but they are somewhat differently defined. Each level aims to keep its maximum load bounded by a parameter. To do this it has the ability to reject jobs. Whenever a job enters the system it is placed in the lowest level that accepts it. A crucial invariant that must be maintained is that if a level rejects a job then a lower bound can placed on the optimum load of the jobs currently in the system. This enables us to obtain a competitive ratio. To derive a reassignment factor we use the method of potential functions. We can ensure that whenever a job has to be reassigned a number of jobs of large reassignment cost have left the system. This means that we can "charge" the reassignment to the jobs that have left.

References

[1] J. Westbrook. Load balancing for response time. In *ESA*, pages 355–368, 1995.