Optimizing Static Job Scheduling in a Network of Heterogeneous Computers*

Xueyan Tang & Samuel T. Chanson
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
E-mail: {tangxy, chanson}@cs.ust.hk

Abstract

This paper investigates static job scheduling schemes that distribute workload in a network of computers with different speeds. Static schemes involve very low overhead and complexity compared to dynamic schemes, but can still provide significant performance improvement over the case of no load balancing. Optimization techniques are proposed for workload allocation and job dispatching. Workload allocation is modeled as a non-linear optimization problem and solved mathematically. It is shown that allocating a disproportionately high percentage of jobs to the more powerful computers improves system performance. The proposed job dispatching algorithm is an extension of the traditional round-robin scheme. The objective is to reduce burstiness in the job arrival stream to each computer. The schemes are evaluated by simulation experiments. Performance results verify their effectiveness in terms of mean response time, mean response ratio, and fairness. The Optimized Round-Robin (ORR) strategy which combines both techniques outperforms other static scheduling algorithms examined.

1 Introduction

Workload distribution is an important factor affecting the performance of a network of computers. Due to uneven job arrival patterns and possible differences in computing capacities, the workload on different computers in the system can vary greatly [1]. Load balancing techniques improve system performance by dispatching jobs from heavily loaded computers to lightly loaded ones. Researchers have proposed a variety of approaches for load balancing. Based on the system information used by the scheduling algorithms, they can be classified into two categories: static and dynamic [16]. Static approaches either do not use any workload information or use only average system behaviors such as mean job arrival rate and execution rate. Unlike the dynamic schemes, they do not consider instantaneous system states in job allocation calculations.

Dynamic schemes usually outperform static schemes. However, they typically involve much higher system overhead (e.g., frequent collection and processing of workload information [12]). For some applications, it may not be necessary to obtain the best possible performance; a healthy performance gain from a good low-cost static algorithm may be more attractive. The objective of this paper is to find ways to optimize static job scheduling schemes. We focus on heterogeneous systems[1], and assume the jobs are independent and do not communicate among themselves or with the users.

A static job scheduling policy typically contains two components: a workload allocation scheme and a job dispatching strategy. The workload allocation scheme determines the fractions of the total workload to be assigned to each computer. The job dispatching strategy decides which jobs are to be sent to each computer as they arrive based on the computed fractions. Optimization techniques for both components are proposed in this paper. A mathematical model is developed to analyze system performance and compute the optimized allocation policy. For job dispatching, a strategy based on the idea of round-robin is presented. It provides smoother job arrival streams seen by individual computers to further improve performance. The effectiveness of the proposed algorithms is studied by simulation.

Optimizing workload allocation for heterogeneous systems is not an easy task. Leslie *et al.* [14] recently showed that distributing workload to computers proportionally to their speeds does not always provide the best performance unless the system load is very high. However, they did not quantitatively investigate how scheduling performance can be optimized. Banawan *et al.* [2] discussed a similar problem, but only an objective function was given without any solution, and they did not consider the job dispatch-

*This work was partially supported by a grant from the Research Grant Council of Hong Kong (Grant No. HKUST6080/97E).

[1]Heterogeneity in this paper refers to differences in computing capacity.

ing strategy. Crovella *et al.* [5] and Schroeder *et al.* [15] studied task assignment policies where task sizes follow a heavy-tailed distribution. They found that system performance can be greatly improved by allocating tasks based on their service demands which results in unbalanced loadings on the servers. Their work assumed task sizes are known *a priori* while this assumption is not needed in our work. Our results are also applicable to the World Wide Web environment. Existing work on domain name server (DNS) scheduling [4] and HTTP request distribution [6] employed simple weighted workload allocation for heterogeneous servers. The performance can be further improved with our proposed optimization techniques.

The rest of the paper is organized as follows. Section 2 provides the theoretical foundation and derives the optimized workload allocation scheme. Section 3 presents a round-robin based job dispatching strategy. Section 4 describes the simulation model and the scheduling algorithms used in performance comparison. Experimental results are discussed in Section 5. Finally, Section 6 concludes the paper.

2 Optimization Technique for Workload Allocation

We model the system as a network of n computers $c_1, c_2, \ldots, c_n$ (see Figure 1). Each computer c_i is associated with a real number $s_i > 0$ which denotes its relative processing speed. Specifically, let μ be the base-line job service rate, then the service rate of c_i is equal to $s_i\mu$. The jobs arrive at the system at an average rate λ. They are assigned to the computers by a central scheduler. The scheduler applies a static job scheduling scheme, where a fraction α_i of all the jobs are sent to computer c_i ($0 \leq \alpha_i \leq 1$ and $\sum_{i=1}^{n} \alpha_i = 1$).

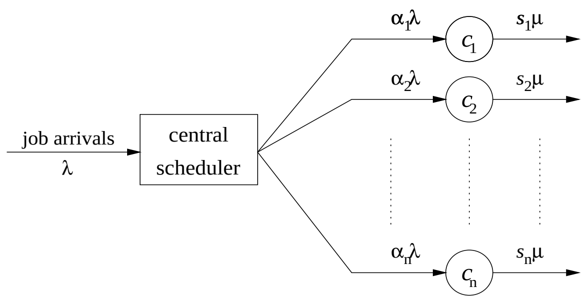


Figure 1. System Model

2.1 Simple Weighted Workload Allocation

A naive workload allocation scheme is to set the amount of workload for each computer proportional to its processing speed. In other words, α_i is set to $\frac{s_i}{\sum_{j=1}^{n} s_j}$. This scheme takes speed heterogeneity into consideration and tries to make all the computers equally utilized. However, it has been shown that it does not provide the best performance.

2.2 Motivation for Optimization

Previous studies [14] have shown that it is beneficial to allocate a disproportionately higher fraction of the workload to the more powerful computers at low and moderate loads, while at high loads, it is better to keep the machines more balanced. Similar phenomena were also observed in our experiments with the Dynamic Least-Load scheduling algorithm in a heterogeneous system [3]. In this algorithm, the central scheduler keeps track of the load (i.e., the run queue length[2]) on each computer. When a new job arrives, it is assigned to the machine with the least normalized load (i.e., $\frac{run\ queue\ length+1}{processing\ speed}$). Table 1 shows an example of the average percentage of jobs allocated to individual computers with different speeds using this algorithm when the overall system utilization is 70%.

speed	1.0	1.5	2.0	3.0
percentage (%)	0.29	1.75	3.84	7.17
speed	5.0	9.0	10.0	
percentage (%)	14.59	27.95	30.90	

Table 1. An Example of Workload Distribution Generated by Dynamic Least-Load Scheduling

Notice that workload is not proportionately divided among the computers according to their speeds. Computers with slow speeds receive much less normalized workload than the fast machines. The distribution of workload becomes even more skewed when the system utilization decreases. An intuitive explanation is as follows. It is known that the response time of a job depends heavily on the utilization of the computer processing it. Consider the case where the workload is distributed proportionately to computer speeds. If some jobs are moved from a slow machine to a fast machine, the utilization of the slow machine will be significantly decreased while that of the fast machine will not increase much due to differences in their capacities. Hence, the job transfer will reduce the response time of jobs on the slow computer without affecting the jobs on the fast computer very much, and the overall system performance will improve.

The observation motivated us to quantitatively analyze the impact of skewed workload allocation and find a better strategy for static job scheduling.

2.3 Optimized Workload Allocation

We model the workload allocation problem as a nonlinear optimization problem and solve it mathematically. The main performance metrics used in our analysis are:

[2]This descriptor has been shown to be a simple and effective load index [13].

1. *Mean response time*: This is defined as the average completion time of all the jobs.

2. *Mean response ratio*: The *response ratio* of a job is defined as the ratio of the job's response time to its size. Job size is the completion time of the job when it is executed on an idle machine with relative speed 1. Mean response ratio is the average response ratio over all the jobs. This metric is more objective as the effect of job size is eliminated.

Every computer is modeled as an $M/M/1$ queue which employs the processor sharing (PS) service discipline at the server [11]. Let T be the job response time, λ be the mean job arrival rate, μ be the mean job service rate and $\rho = \frac{\lambda}{\mu}$ be the server utilization. The expected response time of a job with size t is given by

$$E\{T \mid job\ size = t\} = \frac{t}{1-\rho}.$$

Therefore, the mean response time $\overline{T}$ and the mean response ratio $\overline{R}$ of all the jobs can be calculated as follows:

$$\overline{T} = \frac{1}{1-\rho} \cdot mean\ job\ size = \frac{1}{1-\rho} \cdot \frac{1}{\mu}, \tag{1}$$

and

$$\overline{R} = \frac{1}{1-\rho}. \tag{2}$$

Consider our system model (Figure 1). Let $\overline{T}_i$ and $\overline{R}_i$ be the mean response time and the mean response ratio of the jobs processed by c_i respectively. Since computer c_i receives a fraction α_i of the total workload, equations (1) and (2) can be rewritten as

$$\overline{T}_i = \frac{1}{(1-\frac{\alpha_i\lambda}{s_i\mu})s_i\mu} = \frac{1}{s_i\mu - \alpha_i\lambda},$$

and

$$\overline{R}_i = \frac{1}{1-\frac{\alpha_i\lambda}{s_i\mu}} \cdot \frac{1}{s_i} = \frac{\mu}{s_i\mu - \alpha_i\lambda}.$$

Note that we need to add a factor $\frac{1}{s_i}$ in calculating $\overline{R}_i$ according to the definition of response ratio.

Thus the mean response time $\overline{T}$ and the mean response ratio $\overline{R}$ of the system are

$$\overline{T} = \sum_{i=1}^{n} \frac{1}{s_i\mu - \alpha_i\lambda}\alpha_i, \tag{3}$$

and

$$\overline{R} = \sum_{i=1}^{n} \frac{\mu}{s_i\mu - \alpha_i\lambda}\alpha_i = \mu\overline{T}.$$

Our objective is to find a workload allocation scheme $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ that optimizes system performance. Since μ is a constant, $\overline{R}$ is proportional to $\overline{T}$. Hence, the optimized allocation strategies for $\overline{T}$ and $\overline{R}$ are the same.

Equation (3) can be rewritten as follows:

$$\overline{T} \quad = \quad -\frac{n}{\lambda} + \frac{1}{\lambda}\sum_{i=1}^{n} \frac{s_i\mu}{s_i\mu - \alpha_i\lambda}.$$

Therefore, $\overline{T}$ is minimized if $\sum_{i=1}^{n} \frac{s_i\mu}{s_i\mu-\alpha_i\lambda}$ is minimized.

Definition 1 Let μ be the base-line service rate, λ be the job arrival rate of the system, and $s_1 \le s_2 \le \ldots \le s_n$ be the relative speeds of the computers in the network, where $\lambda < \sum_{i=1}^{n} s_i\mu$ (i.e., the system is not saturated). The objective function $F(\alpha_1, \alpha_2, \ldots, \alpha_n)$ is defined as

$$F(\alpha_1, \alpha_2, \ldots, \alpha_n) = \sum_{i=1}^{n} \frac{s_i\mu}{s_i\mu - \alpha_i\lambda}. \quad \square$$

As a starting point, we minimize the objective function without requiring α_i $(1 \le i \le n)$ to be non-negative.

Theorem 1 The objective function $F(\alpha_1, \alpha_2, \ldots, \alpha_n)$ is minimized when

$$\alpha_i = \frac{1}{\lambda}\Big(s_i\mu - \sqrt{s_i\mu}\frac{\sum_{j=1}^{n} s_j\mu - \lambda}{\sum_{j=1}^{n}\sqrt{s_j\mu}}\Big), \tag{4}$$

subject to the constraints $\sum_{i=1}^{n} \alpha_i = 1$, and $\alpha_i\lambda < s_i\mu$ (the second constraint ensures that no individual computer is saturated). The minimum value of $F(\alpha_1, \alpha_2, \ldots, \alpha_n)$ is

$$\frac{(\sum_{j=1}^{n}\sqrt{s_j\mu})^2}{\sum_{j=1}^{n} s_j\mu - \lambda}.$$

Proof: This is a constrained-minimum problem. It can be solved using the Lagrange multiplier theorem in a straightforward way (see [17] for details). $\square$

The solution provided by Theorem 1 may not always be practical since α_i is not guaranteed to be non-negative. The following theorem is needed to deal with this problem. When α_i calculated in (4) is negative (this means that the speed of computer c_i is very slow, i.e., $s_i\mu < (\frac{\sum_{j=1}^{n} s_j\mu-\lambda}{\sum_{j=1}^{n}\sqrt{s_j\mu}})^2$), an intuitive method to make the solution feasible is to set α_i to 0 and recompute the other fractions again according to Theorem 1. This means we do not assign jobs to the extremely slow computers. Theorem 2 presents this method formally.

Theorem 2 Suppose for an integer m $(1 \le m < n)$, $\sqrt{s_m\mu} < \frac{\sum_{j=m}^{n} s_j\mu-\lambda}{\sum_{j=m}^{n}\sqrt{s_j\mu}}$, then the objective function $F(\alpha_1, \alpha_2, \ldots, \alpha_n)$ is minimized when $\alpha_i = 0$ $(1 \le i \le m)$, subject to the extra constraint $\alpha_i \ge 0$ $(1 \le i \le m)$ in addition to the two constraints stated in Theorem 1.

Proof: See [17] for details. □

In fact, (4) can be rewritten as

$$\alpha_i = s_i\Big(\frac{\mu}{\lambda}\Big) - \sqrt{s_i}\frac{(\frac{\mu}{\lambda})\sum_{j=1}^{n} s_j - 1}{\sum_{j=1}^{n}\sqrt{s_j}},$$

where $\frac{\mu}{\lambda}$ can be derived from the system utilization $\rho = \frac{\lambda}{\mu\sum_{i=1}^{n} s_i}$. As a result, we only need to know ρ and $s_1, s_2, \ldots, s_n$ to compute the optimized allocation strategy $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. The calculation procedure is formally presented in Algorithm 1.

Algorithm 1 Calculation of Optimized Workload Allocation Scheme

```
Input: System Utilization $\rho$; Computer Speeds $s_1, s_2, \ldots, s_n$.
Output: Optimized Allocation Strategy $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$.
  1. Let $\beta = \frac{1}{\rho\sum_{i=1}^{n} s_i}$. ($\beta = \frac{\mu}{\lambda}$)
  2. Sort $s_1, s_2, \ldots, s_n$ in increasing order.
  3. Let $lower = 1$, and $upper = n$.
  4. While ($lower \le upper$)
     (4.a) Let $mid = \lfloor \frac{lower+upper}{2} \rfloor$.
     (4.b) If $\sqrt{s_{mid}\mu} < \frac{\sum_{j=mid}^{n} s_j\mu - \lambda}{\sum_{j=mid}^{n}\sqrt{s_j\mu}}$
              Let $lower = mid + 1$.
     (4.c) Else
              Let $upper = mid - 1$.
  5. Let $m = lower - 1$.
  6. For each $i$ satisfying $1 \le i \le m$
        Let $\alpha_i = 0$.
  7. For each $i$ satisfying $m+1 \le i \le n$
        Let $\alpha_i = s_i\beta - \sqrt{s_i}\frac{\beta\sum_{j=m+1}^{n} s_j - 1}{\sum_{j=m+1}^{n}\sqrt{s_j}}$.
```

The correctness of Algorithm 1 is proved by the following theorem.

Theorem 3 The workload allocation strategy $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ computed in Algorithm 1 minimizes $F(\alpha_1, \alpha_2, \ldots, \alpha_n)$ under the constraints $\sum_{i=1}^{n}\alpha_i = 1$ and $0 \le \alpha_i < \frac{s_i\mu}{\lambda}$ $(1 \le i \le n)$.

Proof: Steps 4 and 5 in Algorithm 1 finds the maximum index $i = m$ such that $\sqrt{s_i\mu} < \frac{\sum_{j=i}^{n} s_j\mu - \lambda}{\sum_{j=i}^{n}\sqrt{s_j\mu}}$ using a binary search[3]. If $m = 0$, it is easy to prove the conclusion using Theorem 1. If $m \ne 0$, for any $1 \le i \le m$, $\sqrt{s_i\mu} < \frac{\sum_{j=i}^{n} s_j\mu - \lambda}{\sum_{j=i}^{n}\sqrt{s_j\mu}}$. Applying Theorem 2, $\alpha_1, \alpha_2, \ldots, \alpha_m$ should all be set to 0 in order to minimize $F(\alpha_1, \alpha_2, \ldots, \alpha_n)$ under the constraints $\alpha_i \ge 0$ $(1 \le i \le m)$ (step 6). Since $\sqrt{s_m\mu} < \frac{\sum_{j=m}^{n} s_j\mu - \lambda}{\sum_{j=m}^{n}\sqrt{s_j\mu}}$, we have $0 < \sqrt{s_m\mu}\sum_{j=m+1}^{n}\sqrt{s_j\mu} < \sum_{j=m+1}^{n} s_j\mu - \lambda$, i.e., $\lambda < \sum_{j=m+1}^{n} s_j\mu$. Therefore, we could apply Theorem 1 to the subset $\{\alpha_{m+1}, \alpha_{m+2}, \ldots, \alpha_n\}$ only, and their values in the optimized allocation strategy that minimizes $F(\alpha_1, \alpha_2, \ldots, \alpha_n)$ are given as follows (step 7):

$$\begin{aligned} \alpha_i &= \frac{1}{\lambda}\Big(s_i\mu - \sqrt{s_i\mu}\frac{\sum_{j=m+1}^{n} s_j\mu - \lambda}{\sum_{j=m+1}^{n}\sqrt{s_j\mu}}\Big) \qquad (5) \\ &= s_i\beta - \sqrt{s_i}\frac{\beta\sum_{j=m+1}^{n} s_j - 1}{\sum_{j=m+1}^{n}\sqrt{s_j}}, \\ &\quad (\text{for any } m+1 \le i \le n). \end{aligned}$$

Since m is the maximum index satisfying $\sqrt{s_i\mu} < \frac{\sum_{j=i}^{n} s_j\mu - \lambda}{\sum_{j=i}^{n}\sqrt{s_j\mu}}$, we have $\sqrt{s_i\mu} \ge \sqrt{s_{m+1}\mu} \ge \frac{\sum_{j=m+1}^{n} s_j\mu - \lambda}{\sum_{j=m+1}^{n}\sqrt{s_j\mu}}$, for any $m+1 \le i \le n$. Thus all the α_i $(m+1 \le i \le n)$ calculated by (5) are guaranteed to be non-negative. Hence, the theorem is proved. □

It can be seen from the analytical result that in the optimized scheme, fast computers receive disproportionately higher share of workload while slow computers are allocated lower proportion or even zero workload. This is consistent with the observations in Section 2.2. The degree of disproportional workload allocation depends on the system utilization. The lower the system load, the more skewed the allocation scheme. When the system utilization approaches 100%, the optimized allocation scheme degenerates to the simple weighted scheme.

3 Optimization Technique for Job Dispatching

The job dispatching strategy is another important component in static job scheduling. Different schemes can be used to split the incoming job stream to the system into n substreams in real time as the jobs arrive, one for each computer in the network proportional to the computed fraction of the total workload for that computer.

3.1 Random Based Job Dispatching

In random based job dispatching, a newly arrived job is scheduled to run on a ”randomly” selected computer where the probability of sending the new job to computer c_i is equal to α_i, given fractions $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ and

[3]It can be proved (see [17]) that the index i's are contiguous and therefore a binary search can be used to locate the maximum index.

$\sum_{i=1}^{n} \alpha_i = 1$. This strategy is straightforward but its performance can vary greatly for different random number sequences.

3.2 Round-Robin Based Job Dispatching

In this subsection, we propose a job dispatching algorithm that tries to improve performance by smoothing the job arrival substream of each computer.

The inter-arrival time between two adjacent jobs sent to the same computer is the sum of several inter-arrival intervals of consecutive jobs in the overall job arrival stream. The objective of the proposed strategy is to equalize the number of original inter-arrival intervals in between successive jobs dispatched for each computer. This has the effect of smoothing out burstiness without having to measure the inter-arrival times. For example, suppose there are four computers c_1, c_2, c_3 and c_4 with workload fractions $\frac{1}{8}, \frac{1}{8}, \frac{1}{4}$ and $\frac{1}{2}$ respectively. An appropriate job dispatching scheme will be $c_4, c_3, c_4, c_2, c_4, c_3, c_4, c_1, c_4, c_3, c_4, c_2, c_4, c_3, c_4, c_1, \ldots$. In this way, the numbers of jobs directed to different computers are proportional to their assigned fractions even in a short interval. When each computer shares the same fraction of workload, this scheme degenerates to the traditional round-robin strategy. Hence, we refer to it as round-robin based job dispatching. Perfectly spreading the jobs as in the above example may not always be possible. In most cases, we can only approximate the equalization as much as possible. A formal description of the round-robin based job dispatching algorithm for the general case is given in Algorithm 2. The algorithm takes the workload fractions from the allocation scheme as input. Each computer is associated with two attributes: the $assign$ field records the number of jobs that have been sent to the computer, and the $next$ field denotes the expected number of incoming jobs before the next job assignment to the computer.

The $assign$ and the $next$ values of every computer are initialized in step 1. Then the algorithm goes into an infinite loop. It sends a newly arrived job to the computer with the minimum expected time (measured in terms of number of incoming jobs) to the next assignment (step 2.c.2). If more than one computer has the same expected time, the one with the smallest number of assigned jobs (including the new job) normalized by its processing speed is selected (step 2.c.3). After making the decision, the expected time of receiving the next job for the selected computer is increased by $\frac{1}{\alpha_{select}}$ (step 2.e). This is because the selected computer is expected to receive one job out of every $\frac{1}{\alpha_{select}}$ arrivals to the system. The number of jobs allocated to the selected computer is then increased by 1 (step 2.f). After dispatching a job to its destination, the $next$ field is decreased by 1 for all computers except those that have not received any job yet (step 2.h).

Algorithm 2 Round-Robin Based Job Dispatching Strategy

Input: Workload fractions from allocation scheme $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, where α_i denotes the fraction of workload assigned to computer c_i, $0 \le \alpha_i \le 1$ $(1 \le i \le n)$ and $\sum_{i=1}^{n} \alpha_i = 1$.

```
1. For $(i = 1;\ i \le n;\ i = i + 1)$ do
    (1.a) Let $c[i].assign = 0$.
    (1.b) Let $c[i].next = 1$.
2. While(true)
    (2.a) Wait until a job arrives.
    (2.b) Let $select = -1$, $minnext = 999999$, and $norassign = -1$.
    (2.c) For $(i = 1;\ i \le n;\ i = i + 1)$ do
        (2.c.1) If $\alpha_i = 0$
                    Continue.
        (2.c.2) Else if $select = -1$ or $minnext > c[i].next$
                    Let $minnext = c[i].next$.
                    Let $norassign = \frac{c[i].assign+1}{\alpha_i}$.
                    Let $select = i$.
        (2.c.3) Else if $minnext = c[i].next$ and $norassign > \frac{c[i].assign+1}{\alpha_i}$
                    Let $norassign = \frac{c[i].assign+1}{\alpha_i}$.
                    Let $select = i$.
    (2.d) If $c[select].assign = 0$
        (2.d.1) Let $c[select].next = 0$.
    (2.e) Let $c[select].next = c[select].next + \frac{1}{\alpha_{select}}$.
    (2.f) Let $c[select].assign = c[select].assign + 1$.
    (2.g) Send the job to computer $c_{select}$.
    (2.h) For $(i = 1;\ i \le n;\ i = i + 1)$ do
        (2.h.1) If $c[i].assign \neq 0$
                    Let $c[i].next = c[i].next - 1$.
```

To decide when a computer should receive its first job, the following algorithm is used. Initially, computers allocated larger fractions of workload are selected first because they have smaller normalized $assign$ values i.e., $\frac{1}{\alpha_i}$ (step 2.c.3). Since the $next$ fields of all computers are initialized to the guard value 1 instead of 0 (step 1), those associated with smaller workload fractions would not be assigned jobs as long as there exists at least one computer with a $next$ value less than 1. This implies that the job arrival patterns for the computers that have started processing jobs are not interrupted. In this way, computers with similar and small workload fractions would receive their first jobs at different times which are evenly spread out in a cycle (e.g., c_1 and

c_2 in the previous example). When a computer is selected for the first time, its $next$ field is reset to 0 (step 2.d) before being updated in the normal way.

As an example, consider 8 computers with workload fractions 0.35, 0.22, 0.15, 0.12, 0.04, 0.04, 0.04 and 0.04 respectively. Assume a hyperexponential job arrival pattern with mean inter-arrival time of 2.2 seconds. Figure 2 shows the workload allocation deviation[4] of round-robin and random based strategies in 30 consecutive intervals where the length of each interval is 120 seconds.

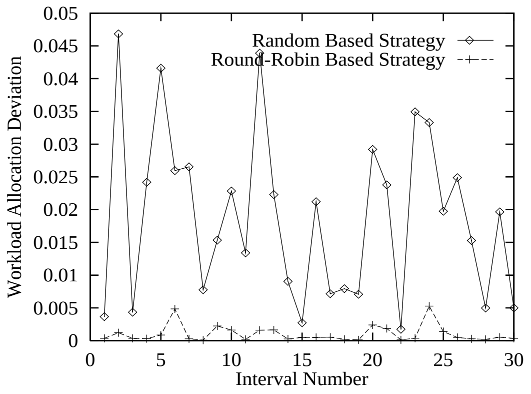


Figure 2. Comparison of Job Dispatching Strategies

It can be seen that the deviations of random based strategy are much higher and fluctuate more widely than those of round-robin based strategy. This implies that the job arrival stream to each computer is less bursty under round-robin based strategy.

The round-robin based job dispatching can be integrated with any workload allocation scheme. Its combinations with the simple weighted and the optimized allocation schemes are referred to as Weighted Round-Robin (WRR) and Optimized Round-Robin (ORR) algorithms respectively. The round-robin based job dispatching strategy does not require information exchange among computers and hence involves little system overhead. Therefore, both WRR and ORR are low-cost job scheduling algorithms.

4 Experimental Setup

4.1 Simulation Model

A discrete-event simulator was developed to evaluate the performance of the proposed job scheduling algorithms. In the simulation model, a collection of computers are connected by a high speed network. A central scheduler receives all incoming jobs and distributes them to the computers according to the specified scheduling algorithm. Once scheduled, the jobs run to completion on the assigned computer and are not rescheduled. The program and data files are assumed to be stored on a dedicated file server. Consequently, the files do not have to be transferred when assigning jobs to remote computers; only a command line is sent. All the computers apply preemptive round-robin processor scheduling.

For simplicity, the theoretical analysis in Section 2.3 was based on the $M/M/1$ queueing model. Our simulation studies have adopted a more realistic workload model. Previous studies [7, 8] have found that job size distributions exhibit a heavy-tailed property in most computing systems. A small number of very large jobs make up a significant fraction of the total load. We used the Bounded Pareto Distribution as the job size distribution in our simulation experiments. The probability density function of the Bounded Pareto Distribution $B(k, p, \alpha)$ is defined as follows [7]:

$$f(x) = \frac{\alpha k^{\alpha}}{1 - (k/p)^{\alpha}} x^{-\alpha-1} \quad (k \le x \le p),$$

where k and p are the lower bound and the upper bound of job size respectively, and α is a parameter that reflects the variability of job size. The default values of these parameters are: $k = 10.0$ seconds, $p = 21600.0$ seconds, and $\alpha = 1.0$. Under this setting, the average job size is 76.8 seconds.

It is known that the job arrival process in a computer system is far from Poisson. The job inter-arrival time of the trace data collected by Zhou [18] has a coefficient of variation (CV) equal to 2.64 while the inter-arrival CV of the Poisson process is 1. This indicates that the instantaneous system load fluctuates greatly. A two-stage hyperexponential distribution [10] is used to model the job arrival process, where the job inter-arrival CV is set to 3.0. The default system utilization is set at 70% in our simulator.

The main performance metrics used in our simulation experiments include *mean response time*, *mean response ratio* (defined in Section 2.2), and *fairness*. Fairness is defined as the standard deviation of the response ratio over all jobs. This definition of fairness is reasonable as users are likely to expect short delays for small tasks and willing to tolerate longer delays for larger tasks. The smaller the fairness, the better the performance.

Each simulation was run for 4.0×10^6 seconds, starting with an idle system. This is sufficient to generate a total of 1 to 2 million jobs. The first quarter of each run (1.0×10^6 seconds) is considered the start up period, allowing the system to get into the steady state. Statistics were collected from the jobs that arrive after the start up period (3.0×10^6 seconds). Each data point shown in the figure is the average result of 10 independent runs with different random number streams.

[4]The workload allocation deviation is defined as $\sum_i (\alpha_i - \alpha_i')^2$, where α_i and α_i' are the expected and actual fractions of jobs allocated to computer c_i in the given interval respectively.

4.2 Job Scheduling Algorithms

In addition to ORR and WRR, two other static scheduling algorithms, Weighted Random (WRAN) and Optimized Random (ORAN) were also considered in the simulation for comparison. WRAN and ORAN are the combinations of random based job dispatching with the simple weighted and the optimized workload allocation respectively. WRAN is the simplest static scheduling algorithm that takes computer speed into consideration. The performance difference between ORR and WRAN provides a suitable estimate of the effectiveness of ORR. Essentially, the four static scheduling algorithms studied are various combinations of job dispatching strategies and workload allocation schemes. Their relationships are summarized in Table 2.

		workload allocation scheme	
		weighted	optimized
job dispatching strategy	random	WRAN	ORAN
	round-robin	WRR	ORR

Table 2. Combinations of Job Dispatching and Workload Allocation Schemes

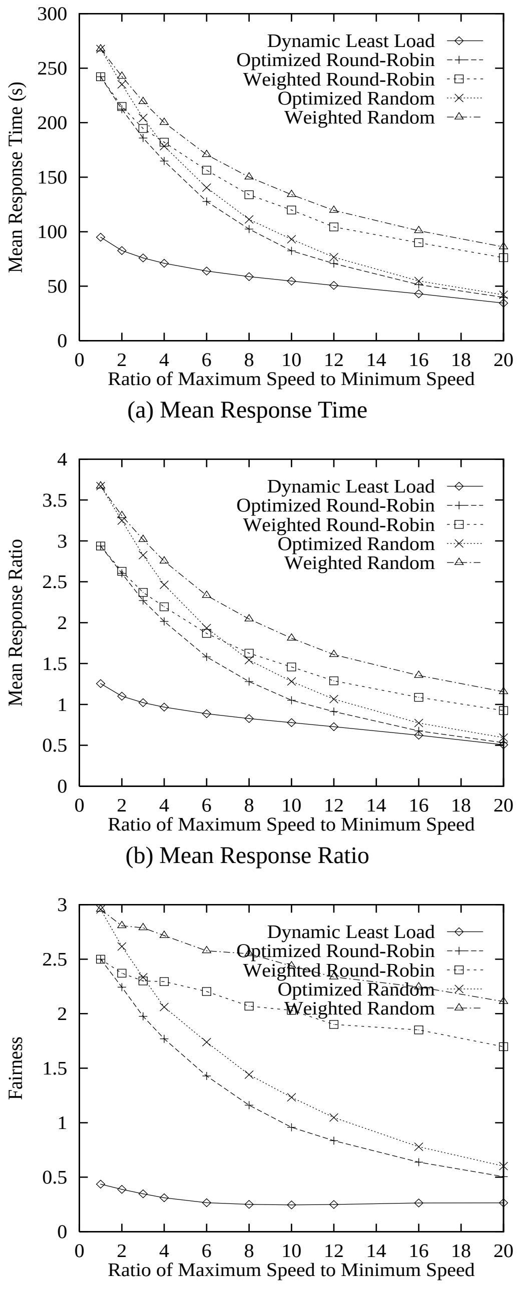
We also included the Dynamic Least-Load algorithm (see Section 2.2) in the study. It is used as a yardstick (upper bound) on the performance of the static schemes. In this algorithm, the load index of a computer is updated in two situations: job arrival and job departure. For job arrival, since no job rescheduling is allowed, the scheduler updates the load index of the target computer immediately after sending the job [9]. For job departure, the load update process has to be initiated by the computer that had just executed the job. We assume that each computer checks its load index every second. Therefore, after a job is completed on a computer, it takes the computer $U(0, 1)$ second to detect the load change, where $U(x, y)$ is a uniformly distributed number between x and y. Then the computer sends a load update message to the scheduler. The message transfer delay is set to be exponentially distributed with some mean value (currently set at 0.05 second).

5 Performance Evaluation

The job scheduling algorithms mentioned in Section 4.2 were evaluated under a wide range of system configurations and workload levels. The performance results are presented in this section.

5.1 Effect of Speed Skewness

The optimized workload allocation scheme is designed for a group of computers with different speeds. In this subsection, we examine its effectiveness by varying the speeds of the computers. The system consists of 18 computers which are divided into two classes: 2 fast computers and 16 slow computers. The speed of the slow computers is fixed at 1 and the speed of the fast computers is varied from 1 to 20, ranging from a homogeneous system to a highly skewed system. Figure 3 presents the performance results.



(a) Mean Response Time

(b) Mean Response Ratio

(c) Fairness

Figure 3. Performance of Different Skewness of Computer Speeds

Strategies using optimized workload allocation (ORR and ORAN) reduce mean response time and mean response ratio over strategies employing simple weighted allocation

(WRR and WRAN) when the system is not homogeneous. The performance difference increases as the speed differential increases. When the speed ratio between the two types of computers is 20:1, ORR outperforms WRR by 42% and ORAN outperforms WRAN by 49% in terms of mean response ratio. The reason is that the optimized allocation strategy emulates Dynamic Least-Load by sending a higher proportion of workload to the more powerful computers while the simple weighted strategy does not make full use of speed skewness. The performance of ORR and ORAN approaches that of Dynamic Least-Load when the speed of the fast computers grows beyond 20 or so. Moreover, it can be seen from Figure 3(c) that ORR and ORAN exhibit much better fairness than WRR and WRAN. This implies that the response time of a job is more predictable under an optimized allocation strategy. A number of experiments for different system configurations have been carried out and they all show the same performance trends[5]. Thus, we conclude that it is beneficial to apply the optimized workload allocation strategy when the system consists of fast and slow machines.

Another observation is that ORR and WRR perform better than ORAN and WRAN respectively. This verifies that by reducing burstiness in the job arrival stream seen by each computer, round-robin based job dispatching outperforms random based dispatching.

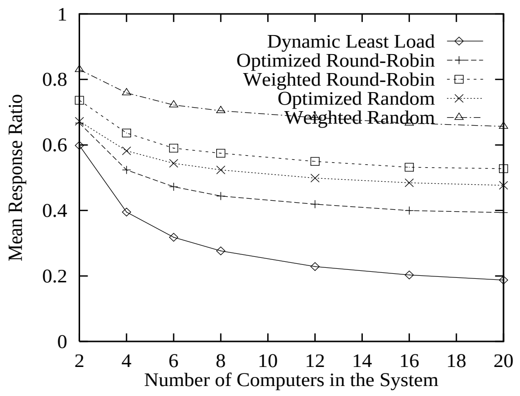
It is also interesting to compare the relative significance of optimizing workload allocation and improving job dispatching (see Figure 3). When the system is close to homogeneous, applying round-robin based job dispatching shows greater advantage. We can see that WRR performs better than ORAN in this case, and optimization of workload allocation does not do much to improve performance. On the other hand, when the speeds of the machines are very different, the performance of WRR is not as good as ORAN. This indicates that optimizing workload allocation is more important in a highly skewed system. ORR integrates both techniques (optimized workload allocation and round-robin dispatching) and outperforms other static scheduling algorithms studied.

5.2 Effect of System Size

In this set of experiments, the computers are divided into equal numbers of fast and slow machines. The processing speeds of the slow and the fast computers are set to 1 and 10 respectively. Figure 4 shows the performance of the algorithms[6] when the number of computers increases from 2 to 20.

[5]The figures are not presented here due to space limitation, see [17] for details.

[6]Due to space limitation, the result of mean response time is not shown in this and subsequent subsections as the performance trends of mean response time are similar to those of mean response ratio.



(a) Mean Response Ratio

Figure 4. Performance of Different System Sizes

ORR reduces mean response ratio over WRAN by 35% to 40% when there are more than 6 computers. The performance gain of optimizing workload allocation is maintained when system size increases. This is consistent with the analytical models used to calculate the optimized and simple weighted allocation schemes. However, the performance difference between ORR and Dynamic Least-Load increases as the system grows in size. The reason is that ORR simply allocates the same fraction of jobs to computers with the same speed and does not dynamically balance their loads. In contrast, Dynamic Least-Load makes use of instantaneous load information and hence performs better when there are more computers.

For job dispatching, the performance of algorithms using round-robin based dispatching (ORR and WRR) improves as the size of the system grows. This is because the inter-arrival time of jobs at each computer shows less variability with increasing number of computers. By contrast, algorithms using random based dispatching (ORAN and WRAN) are not able to smooth out the burstiness as much and show worse performance than round-robin based strategies.

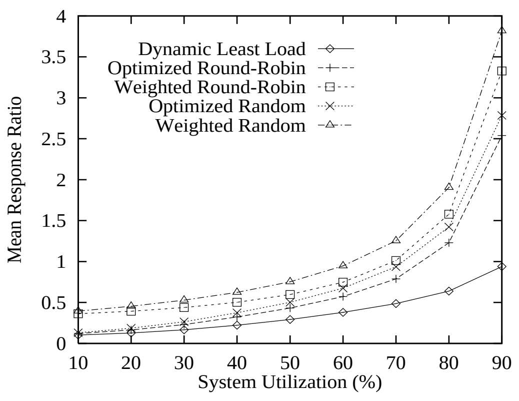
5.3 Effect of System Load

The following system setting (see Table 3) is used as the base configuration in this and subsequent subsections. It consists of 15 computers with six different speeds and the aggregate processing speed is 44.

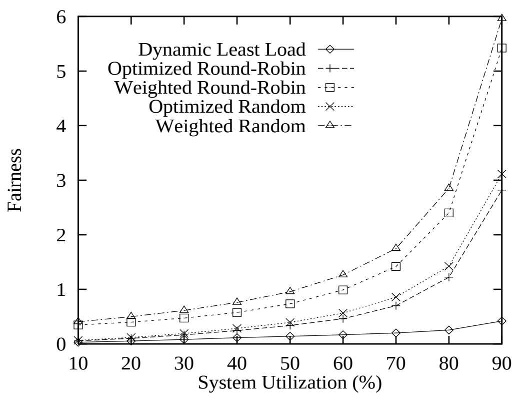
speed	number	speed	number	speed	number
1.0	5	2.0	3	10.0	1
1.5	4	5.0	1	12.0	1

Table 3. Base System Configuration

Figure 5 shows the performance results. It can be seen that ORR outperforms other static algorithms studied. At low and moderate loads, strategies applying optimized



(a) Mean Response Ratio



(b) Fairness

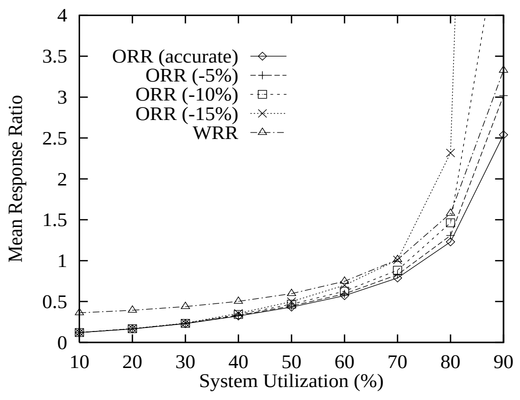
Figure 5. Performance of Different System Utilizations

workload allocation (ORR and ORAN) schedule most of the jobs to fast computers. Their performances are significantly better than those using simple weighted allocation, and are close to that of Dynamic Least-Load. At high loads, ORR and ORAN still exhibit considerable improvement over WRR and WRAN. For example, at a load level of 90%, the mean response ratio of ORR is 24% less than WRR and 34% less than WRAN. Moreover, ORR and ORAN perform much better in fairness than simple weighted schemes (WRR and WRAN). Thus, it can be concluded that proper adjustment in workload allocation greatly enhances system performance over a wide range of system loads. The performance difference between ORR and Dynamic Least-Load increases under very heavy loadings. This implies that dynamic load balancing, which involves high system overhead, is mostly needed when the system utilization is high.

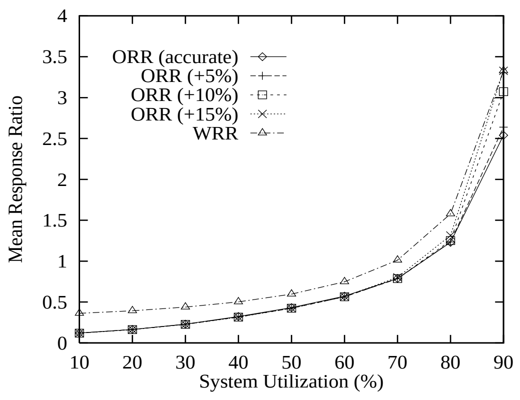
Regarding job dispatching, round-robin based schemes maintain a measurable advantage over random-based schemes in all performance metrics examined. The performance gain is higher under heavy load. This is because the burstiness in job arrivals does little harm when system utilization is low, and system performance becomes more sensitive to job arrival pattern when system load is high.

5.4 Sensitivity to Load Estimation

System utilization is needed in calculating the optimized workload allocation strategy. In this section, we assess the impact of incorrect load estimation on ORR. Figure 6(a) and (b) show the performance when system utilization is underestimated and overestimated respectively. The numbers in the brackets represent the relative estimation errors. For example, ORR(+5%) means that a utilization level of $1.05 \times \rho$ is assumed in the calculation, where ρ is the correct system load.



(a) Underestimation



(b) Overestimation

Figure 6. Performance Sensitivity to Load Estimation

It can be observed from Figure 6(a) that underestimation of load does not affect the performance much when the load is light. However, when the system utilization is high, the advantage of ORR is offset by load underestimation, especially when the error is large. Highly inaccurate information may even cause ORR to perform worse than WRR and make the system unstable. This is because underestimation of load will cause too much workload to be assigned to the fast computers and overload them. In contrast, Figure 6(b)[7] shows that the ORR algorithm is relatively insensi-

[7]The performance result of WRR is adopted for ORR(+15%) at the load level of 90%, since ORR converges with WRR as utilization approaches 100%.

tive to load overestimation. The mean response ratio is only marginally worse up to 80% system utilization and measurable performance gain is maintained at 90% utilization with a +10% estimation error. The reason is that overestimation of system load makes the optimized workload allocation scheme more conservative and closer to the simple weighted scheme. Therefore, it does little harm to the performance.

In summary, the performance of optimized allocation scheme is less sensitive to overestimation than underestimation of system load. We suggest to conservatively overestimate system load slightly for the practical use of ORR.

Notice that in our simulation experiments, the instantaneous system load fluctuates greatly (job inter-arrival coefficient of variance is set at 3, see Section 4.1). Therefore, the results reported in Section 5 would indicate that using the average system utilization over a long period of time is sufficient. It is not necessary to measure ρ and recompute the optimized workload allocation strategy often.

6 Conclusion

We have presented two optimization techniques for static job scheduling in a computer network consisting of machines with different processing speeds. These techniques involve little system overhead compared to the dynamic schemes which require frequent collection and processing of workload information on all machines. The key idea of optimizing the workload allocation scheme is to send a disproportionately high fraction of workload to the more powerful computers. An analytical model is developed to derive the optimized allocation strategy mathematically. For job dispatching, an algorithm that extends the idea of round-robin to the general case is presented. The objective is to reduce the burstiness in the job arrival stream for each computer. Simulation results show that the Optimized Round-Robin (ORR) algorithm which integrates these two techniques outperforms other static scheduling schemes examined. The performance gain increases as the differences between the computer speeds increase. These optimization techniques can be applied to other resource management problems (e.g., replicated heterogeneous server selection) where the resources have different capacities.

References

[1] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW team. A case for now (networks of workstations). *IEEE Micro*, 15(1):54–64, Feb. 1995.

[2] S. A. Banawan and N. M. Zeidat. A comparative study of load sharing in heterogeneous multicomputer systems. In *Proceedings of the 25th Annual Simulation Symposium*, pages 22–31, Apr. 1992.

[3] S. T. Chanson, W. Deng, C.-C. Hui, X. Tang, and M. Y. To. Multidomain load balancing. Technical Report HKUST-CS99-18, Department of Computer Science, HKUST, Dec. 1999.

[4] M. Colajanni, P. S. Yu, and V. Cardellini. Dynamic load balancing in geographically distributed heterogeneous web servers. In *Proceedings of the 18th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 295–302, May 1998.

[5] M. E. Crovella, M. Harchol-Balter, and C. D. Murta. Task assignment in distributed system: Improving performance by unbalancing load. In *Proceedings of the 1998 ACM SIGMETRICS and IFIP Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'98/PERFORMANCE'98)*, pages 268–269, June 1998.

[6] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available web server. In *Proceedings of the 41st IEEE Computer Society International Conference (COMPCON)*, pages 85–92, Feb. 1996.

[7] M. Harchol-Balter, M. E. Crovella, and C. D. Murta. On choosing a task assignment policy for a distributed server system. *Journal of Parallel and Distributed Computing*, 59(2):204–228, Nov. 1999.

[8] M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Transactions on Computer Systems*, 15(3):253–285, Aug. 1997.

[9] C.-C. Hui and S. T. Chanson. Improved strategies for dynamic load balancing. *IEEE Concurrency*, 7(3):58–67, July–September 1999.

[10] L. Kleinrock. *Queueing Systems, Volume I: Theory*. John Wiley & Sons, 1975.

[11] L. Kleinrock. *Queueing Systems, Volume II: Computer Applications*. John Wiley & Sons, 1975.

[12] P. Krueger and N. G. Shivaratri. Adaptive location policies for global scheduling. *IEEE Transactions on Software Engineering*, 20(6):432–444, June 1994.

[13] T. Kunz. The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions on Software Engineering*, 17(7):725–730, July 1991.

[14] R. Leslie and S. McKenzie. Evaluation of loadsharing algorithms for heterogeneous distributed systems. *Computer Communications*, 22(4):376–389, Mar. 1999.

[15] B. Schroeder and M. Harchol-Balter. Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, Aug. 2000. To appear.

[16] N. G. Shivaratri, P. Krueger, and M. Singhal. Load distribution for locally distributed systems. *IEEE Computer*, 8(12):33–44, Dec. 1992.

[17] X. Tang and S. T. Chanson. Optimizing static job scheduling in a network of heterogeneous computers. In *Proceedings of the 29th International Conference on Parallel Processing (ICPP)*, pages 373–382, Aug. 2000.

[18] S. Zhou. A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering*, 14(9):1327–1341, Sept. 1988.