Scheduling with Equipartition

2000; Edmonds

Jeff Edmonds
York University, Toronto, ON, Canada

Synonyms

Round Robin and Equi-partition are the same algorithm. Average Response time and Flow are basically the same measure.

Problem Definition

The task is to schedule a set of n on-line jobs on p processors. The jobs are $J = \{J_1, \ldots, J_n\}$ where *job* J_i has a *release/arrival time* r_i and a sequence of phases $\langle J_i^1, J_i^2, \ldots, J_i^{q_i} \rangle$. Each phase is represented by $\langle w_i^q, \Gamma_i^q \rangle$, where w_i^q denotes the amount of *work* and Γ_i^q is the *speedup function* specifying the rate $\Gamma_i^q(\beta)$ at which this work is executed when given β processors.

A phase of a job is said to be *fully parallelizable* if its speedup function is $\Gamma(\beta) = \beta$. It is said to be *sequential* if its speedup function is $\Gamma(\beta) = 1$.[1] A speedup function Γ is *nondecreasing* iff $\Gamma(\beta_1) \le \Gamma(\beta_2)$ whenever $\beta_1 \le \beta_2$;[2] is *sublinear* iff $\Gamma(\beta_1)/\beta_1 \ge \Gamma(\beta_2)/\beta_2$;[3] and is *strictly-sublinear* by α iff $\Gamma(\beta_2)/\Gamma(\beta_1) \le (\beta_2/\beta_1)^{1-\alpha}$.

An *s-speed scheduling algorithm* $S_s(J)$ allocates $s \times p$ processors each point in time to the jobs J in a way such that all the work completes.[4] More formally, it constructs a function $\mathcal{S}(i,t)$ from $\{1, \ldots, n\} \times [0, \infty)$ to $[0, sp]$ giving the number of processors allocated to job J_i at time t. (A job is allowed to be allocated a non-integral number of processors.) Requiring that for all t, $\sum_{i=1}^{n} \mathcal{S}(i,t) \le sp$ ensures that at most sp processors are allocated at any given time. Requiring that for all i, there exist $r_i = c_i^0 < c_i^1 < \cdots < c_i^{q_i}$ such that for all $1 \le q \le q_i$, $\int_{c_i^{q-1}}^{c_i^q} \Gamma_i^q(\mathcal{S}(i,t))dt = w_i^q$ ensures that before a phase of a job begins, the job must have been released and all of the previous phases of the job must have completed. The *completion* time of a job J_i, denoted c_i, is the completion time of the last phase of the job.

The goal of a scheduling algorithm is to minimize the *average response time*, $\frac{1}{n}\sum_{i \in J}(c_i - r_i)$, of the jobs or equivalently its *flow time* $S_s(J) = \sum_{i \in J}(c_i - r_i)$. An alternative formalization is to integrate over time the number of jobs n_t alive at time t, $S_s(J) = \sum_{i \in J} \int_0^\infty (J_i \text{ is alive a time } t)\delta t = \int_0^\infty n_t \delta t$.

A scheduling algorithm is said to be *on-line* if it lacks knowledge of which jobs will arrive in the future. It is said to be *non-clairvoyant* if it also lacks all knowledge about the jobs that are currently in the system, except for knowing when a job arrives and knowing when it completes.

The two examples of *non-clairvoyant* schedulers that are often used in practice are *Equi-partition* (also called *Round Robin*) and *Balance*. *EQUI*$_s$ is defined to be the scheduler that allocates an equal number of processors to each job that is currently alive. That is, for all i and t, if job J_i is alive at time t, then $\mathcal{EQUI}(i,t) = sp/n_t$, where n_t is the number of jobs that are alive at time t. The schedule *BAL*$_s$ is defined in [8] to be the schedule that allocates all of its processors to the job that has been allocated processors for the shortest length of time. (Though no one implements Balance directly, Unix uses a multi-level feedback (MLF) queue algorithm which in a way approximates Balance).

The most obvious worst-case measure of the goodness of an online non-clairvoyant scheduling algorithm S is its *competitive ratio*. This compares the perform of the algorithm to that of the optimal scheduler. However, in many cases, the limited algorithm is unable to compete against an all knowing all powerful optimal scheduler. To compensate the algorithm S_s, it is given extra speed s. An online scheduling algorithm S is said to be *s-speed c-competitive* if: $\max_J S_s(J)/Opt(J) \le c$. For example, being s-speed 2-competitive means that the cost $S_s(J)$ of scheduler S with $s \times p$ processors on any instance J is at most twice the optimal cost for the same jobs when only given p processors.

Key Results

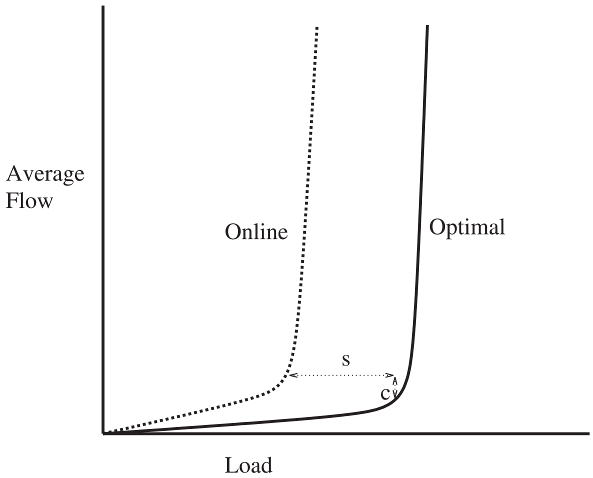
If all jobs arrive at time zero (batch), then the flow time of *EQUI* is 2-competitive on fully parallelizable jobs [10] and $(2 + \sqrt{3})$-competitive on jobs with nondecreasing sublinear speedup functions [3]. (The time until the last job completes (makespan) on fully parallelizable jobs is the same for *EQUI* and *OPT*, but can be a factor of $\Theta(\log n / \log\log n)$ worse for *EQUI* if the jobs can also have sequential phases [11].) Table 1 summarizes all the results.

[1] Note that an odd feature of this definition is that a sequential job completes work at a rate of 1 even when absolutely no processors are allocated to it. This assumption makes things easier for the adversary and harder for any non-clairvoyant algorithm. Hence, it only makes these results stronger.

[2] A job phase with a nondecreasing speedup function executes no slower if it is allocated more processors.

[3] A measure of how efficient a job utilizes its processors is $\Gamma(\beta)/\beta$, which is the work completed by the job per time unit per processor. A sublinear speedup function is one whose efficiency does not increase with more processors. This is a reasonable assumption if in practice β_1 processors can simulate the execution of β_2 processors in a factor of at most β_2/β_1 more time.

[4] $S^s(J)$ is defined to be the scheduler with p processors of speed s. S_s and S^s are equivalent on fully parallelizable jobs and S^s is s times faster than S_s on sequential jobs.



Scheduling with Equipartition, Figure 1
To understand the motivation for this *resource augmentation analysis* [8], note that it is common for the quality of service of a system to have a *threshold property* with respect to the load that it is given. In this example, it seems that the online scheduling algorithm *S* performs reasonably well in comparison to the optimal scheduling algorithm. Despite this, one can see that the competitive ratio of *S* is huge by looking at the vertical gap between the curves when the load is near capacity. To explain why these curves are close, one must also measure the horizontal gap between curves. *S* performs at most *c* times worse than optimal, when either the load is decreased or equivalently the speed is increased by factor of *s*

When the jobs have arbitrary arrival times and are fully parallelizable, the optimal schedule simply allocates all the processors to the jobs with least remaining work. This, however, requires the scheduler to know the amount of work per job. Without this knowledge, *EQUI* and *BAL* are unable to compete with the optimal and hence can do a factor of $\Omega(n/\log n)$ and $\Omega(n)$ respectively worse and no non-clairvoyant schedulers has a better competitive ratio than $\Omega(n^{1/3})$ [9,10]. Randomness improves the competitive ratio of *BAL* to $\Theta(\log n \log\log n)$ [7]. Having more (or faster) processors also helps. BAL_s achieves a $s = 1+\epsilon$ speed competitive ratio of $\frac{s}{s-1} = 1 + \frac{1}{\epsilon}$ [8].

If some of the jobs are fully parallelizable and some are sequential jobs, it is hard to believe that any non-clairvoyant scheduler, even with sp processors, can perform well. Not knowing which jobs are which, it waists too many processors on the sequential jobs. Being starved, the fully parallelizable jobs fall further and further behind and then other fully parallelizable jobs arrive which fall behind as well. For example, even the randomized version of *BAL* can have an arbitrarily bad competitive ratio, even when given arbitrarily fast processors.

EQUI, however, does amazingly well. $EQUI_s$ achieves a $s = 2+\epsilon$ speed competitive ratio of $2 + \frac{4}{\epsilon}$ [1]. This was later improved to $1 + \mathcal{O}(\frac{\sqrt{s}}{s-2})$, which is better for large s [1]. The intuition is that $EQUI_s$ is able to automatically "self adjust" the number of processors wasted on the sequential jobs. As it falls behind, it has more uncompleted jobs in the system and hence allocates fewer processors to each job and hence each job utilizes the processors that it is given more efficiently. The extra processors are enough to compensate for the fact that some processors are still wasted on sequential jobs. For example, suppose the job set is such that *OPT* has ℓ_t sequential jobs and at most one fully parallelizable job alive at any point in time t. (The proof starts by proving that this is the worst case.) It may take a while for the system under $EQUI_s$ to reach a "steady state", but when it does, m_t, which denotes the number of fully parallelizable jobs it has alive at time t, converges to $\frac{\ell_t}{s-1}$. At this time, $EQUI_s$ has $\ell_t + m_t$ jobs alive and *OPT* has $\ell_t + 1$. Hence, the competitive ratio is $EQUI_s(J)/OPT(J) = (\ell_t + \frac{\ell_t}{s-1}))/(\ell_t + 1) \le \frac{s}{s-1}$ CE2, which is $1 + \frac{1}{e}$ for $s = 1+\epsilon$. This intuition makes it appear that speed $s = 1+\epsilon$ is sufficient. However, unless the speed is at least 2 then the competitive ratio can be bad during the time until it reaches this steady state, [8].

More surprisingly if all the jobs are *strictly sublinear*, i. e., are not fully parallel, then *EQUI* performs competitively with no extra processors [1]. More specifically, it is shown that if all the speedup functions are no more fully parallelizable than $\Gamma(\beta) = \beta^{1-\alpha}$ than the competitive ratio is at most $2^{\frac{1}{\alpha}}$. For intuition, suppose the adversary allocates $\frac{p}{n}$ processors to each of n jobs and *EQUI* falls behind enough so that it has $2^{\frac{1}{\alpha}} n$ uncompleted jobs. Then it allocates $p/(2^{\frac{1}{\alpha}} n)$ processors to each, completing work at an overall rate of $(2^{\frac{1}{\alpha}} n)\Gamma(p/(2^{\frac{1}{\alpha}} n)) = 2 \cdot n\Gamma(p/n)$. This is a factor of 2 more than that by the adversary. Hence, as in the previous result, *EQUI* has twice the speed and so performs competitively.

The results for $EQUI_s$ can be extended further. There is a competitive $s = (8+\epsilon)$-speed non-clairvoyant scheduler that only preempts when the number of jobs in the system goes up or down by a factor of two (in some sense $\log n$ times). There is $s = (4+\epsilon)$-speed one that includes both sublinear and superlinear jobs. Finally, there is a $s = \mathcal{O}(\log p)$ speed one that includes both nondecreasing and gradual jobs.

The proof of these results for $EQUI_s$ require techniques that are completely new. For example, the previous results prove that their algorithm is competitive by proving that at every point in time, the number of jobs alive under their al-

CE2 Unbalanced parantheses. Please check.

Editor's or typesetter's annotations (will be removed before the final TEX run)

Scheduling with Equipartition, Table 1

Each row represents a specific scheduler and a class $\mathcal{J}$ of job sets. Here $EQUI_s$ denotes the Equi-partition scheduler with s times as many processors and $EQUI^s$ the one with processors that are s times as fast. The graphs give examples of speedup functions from the class of those considered. The columns are for different extra resources ratios s. Each entry gives the corresponding ratio between the given scheduler and the optimal

	$s = 1$	$s = 1 + \epsilon$	$s = 2 + \epsilon$	$s = 4 + 2\epsilon$	$s = \mathcal{O}(\log p)$
Batch , , or	$[2.71, 3.74]$				
Det. Non-clair	$\Omega(n^{\frac{1}{3}})$	–			
Rand. Non-clair	$\widetilde{\Theta}(\log n)$	–			
Rand. Non-clair or	$\Omega(n^{\frac{1}{2}})$	$\Omega(\frac{1}{\epsilon})$			
BAL_s	$\Omega(n)$	$1 + \frac{1}{\epsilon}$	$\frac{2}{s}$		
BAL_s	$\Omega(s^{-1/\alpha} n)$				
$EQUI_s$, , or	$\Omega(\frac{n}{\log n})$	$\Omega(n^{1-\epsilon})$	$[1 + \frac{1}{\epsilon}, 2 + \frac{4}{\epsilon}]$	≥ 1	
$EQUI^s$, , or	$\Omega(\frac{n}{\log n})$	$\Omega(n^{1-\epsilon})$	$[\frac{2}{3}(1 + \frac{1}{\epsilon}), 2 + \frac{4}{\epsilon}]$	$[\frac{2}{s}, \frac{16}{s}]$	
$EQUI$ or	$[1.48^{1/\alpha}, 2^{1/\alpha}]$				
$EQUI'_s$ Few Preempts			$\Omega(n^{1-\epsilon})$	$\Theta(1)$	
$H\,EQUI_s$ or			$\Omega(n^{1-\epsilon})$	$\Theta(1)$	
$H\,EQUI'_s$ or	$\Omega(n)$				$\Theta(1)$

gorithm is within a constant fraction of that under the optimal schedule. This, however, is simply not true with this less restricted model. There are job sets such that for a period of time the ratio between the numbers of alive jobs under the two schedules is unbounded. Instead, a potential function is used to prove that this can only happen for a relatively short period of time.

The proof first transforms each possible input into a canonical input that as described above only has parallelizable or sequential phases. Having the number of fully parallelizable jobs alive under $EQUI_s$ at time t be much bigger than the number of sequential jobs alive at this same time is bad for $EQUI_s$ because it then has many more jobs alive then OPT and hence is currently incurring much higher costs. On the other hand, this same situation is also good for $EQUI_s$ because it means that it is allocating a larger fraction of its processors to the fully parallelizable jobs and hence is catching up to OPT. Both of these aspects of the current situation is carefully measured in a potential function $\Phi(t)$. It is proven that at each point in time, the occurred cost to $EQUI_s$ plus the gain $\frac{d\Phi(t)}{dt}$ in this potential function is at most c times the costs occurred by OPT. Assuming that the potential function begins and ends at zero, the result follows.

More formally, the potential function is $\Phi(t) = F(t) + Q(t)$ where $Q(t)$ is total sequential work finished by $EQUI_s$ by time t minus the total sequential work finished by the adversary by time t. To define $F(t)$ requires some preliminary definitions. For $u \geq t$, define $m_u(t)$ $(\ell_u(t))$ to be number of fully parallelizable (sequential) phases executing under $EQUI_s$ at time u, for which $EQUI_s$ at time u has still not processed as much work as the adversary processed at time t. Let $n_u(t) = m_u(t) + \ell_u(t)$. Then $F(t) = \int_t^\infty f_u(m_u(t), \ell_u(T))du$, where $f_u(m, \ell) = \frac{s}{s-2} \frac{(m-\ell)(m+\ell)}{n_u}$. As the definition of the potential function suggests, the analysis is quite complicated.

Applications

In addition to being interesting results on their own, they have been powerful tools for the theoretical analysis of other on-line algorithms. For example, in [2,4] TCP was reduced to this problem and in [5], the online broadcast scheduling problem was reduced to this problem.

Open Problems

An open question is whether there is an algorithm that is competitive when given processors of speed $s = 1 + \epsilon$ (as opposed to $s = 2 + \epsilon$). There is a candidate algorithm that is part way between $EQUI_s$ and BAL_s.

Cross References

► Flow Time Minimization
► List Scheduling
► Minimum Flow

- ► Minimum Weighted Completion Time
- ► Online List Update
- ► Online Load Balancing
- ► Schedulers for Optimistic Rate Based Flow Control
- ► Shortest Elapsed Time First Scheduling

Recommended Reading

1. Edmonds, J.: Scheduling in the dark. Improved results: manuscript 2001. In: Theor. Comput. Sci. **235**, 109–141 (2000). In: 31st Ann. ACM Symp. on Theory of Computing, 1999
2. Edmonds, J.: On the Competitiveness of AIMD-TCP within a General Network. In: CE3 LATIN, Latin American Theoretical Informatics, vol. 2976, pp. 577–588 (2004). Submitted to Journal Theoretical Computer Science and/or Lecture Notes in Computer Science CE4
3. Edmonds, J., Chinn, D., Brecht, T., Deng, X.: Non-clairvoyant Multiprocessor Scheduling of Jobs with Changing Execution Characteristics. In: 29th Ann. ACM Symp. on Theory of Computing, 1997, pp. 120–129. Submitted to SIAM J. Comput. CE5
4. Edmonds, J., Datta, S., Dymond, P.: TCP is Competitive Against a Limited Adversary. In: SPAA, ACM Symp. of Parallelism in Algorithms and Achitectures, 2003, pp. 174–183
5. Edmonds, J., Pruhs, K.: Multicast pull scheduling: when fairness is fine. Algorithmica **36**, 315–330 (2003)
6. Edmonds, J., Pruhs, K.: A maiden analysis of longest wait first. In: Proc. 15th Symp. on Discrete Algorithms (SODA) CE6
7. Kalyanasundaram, B., Pruhs, K.: Minimizing flow time nonclairvoyantly. In: Proceedings of the 38th Symposium on Foundations of Computer Science, October 1997
8. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. In: Proceedings of the 36th Symposium on Foundations of Computer Science, October 1995, pp. 214–221
9. Matsumoto, CE7: Competitive Analysis of the Round Robin Algorithm. in: 3rd International Symposium on Algorithms and Computation, 1992, pp. 71–77
10. Motwani, R., Phillips, S., Torng, E.: Non-clairvoyant scheduling. Theor. Comput. Sci. **130** (Special Issue on Dynamic and On-Line Algorithms), 17–47 (1994). Preliminary Version in: Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, 1993, pp. 422–431
11. Robert, J., Schabanel, N.: Non-Clairvoyant Batch Sets Scheduling: Fairness is Fair enough. Personal Correspondence (2007)

CE3 Please provide editor.
CE4 Please clarify and update.
CE5 Please update.
CE6 Please provide location and date of the symposium.
CE7 Please provide initials of "Matsumoto"