Dynamic Coordination Mechanisms

Janardhan Kulkarni †
Vahab Mirrokni ‡

ABSTRACT

Handling the lack of coordination while designing efficient algorithms in distributed systems has been a major topic of study in the past decade. Coordination mechanisms have been proposed as a tool to deal with the issue as well as lack of access to global information in settings such as distributed systems. In the context of resource allocation, a coordination mechanism is a set of local policies that assigns a cost to each strategy based on the available local information. For example, in machine scheduling, this cost only depends on the processing times of jobs assigned to the same machine. Although a great tool to study distributed algorithms in the presence of self-interested agents, coordination mechanisms have few deficiencies as an analysis tool for distributed game theoretic environments. For example, in many real-world settings, we do not know the exact processing time of a job before it finishes. Furthermore, in many settings, jobs arrive online, and have different release times. Motivated by these requirements, we propose dynamic coordination mechanisms, in which each job selects a machine by looking at the set of jobs currently on each machine and it can change its decision over time. In other words, jobs can dynamically choose the (best machine dynamically). We study scheduling and resource allocation problems in this framework.

Here, we are given a set of $\mathcal{M}$ machines and a set $\mathcal{N}$ of jobs or players. Each job $j \in \mathcal{N}$ has $p_j$ units of processing requirement. The mechanism designer, however, is not aware of the processing lengths of jobs. This is commonly referred to as non-clairvoyant setting in the machine scheduling literature. We consider two machine models: In the related machine model, each machine has a speed $\nu_i$, and a job can choose any machine $i \in \mathcal{M}$. In the restricted assignment model, every machine has same speed; however, a job $j$ can only go to a subset of machines $\mathcal{M}_j$. The goal of system designer is to design policies that determine how each machine distributes the total speed $\nu_i$ among the set of jobs which choose the machine $i$. We demand that these policies be local. The jobs leave the system once they complete.

Therefore, a strategy for a job at each time instant consists of choosing a single machine $i \in \mathcal{M}$ that gives it the highest speed so that its processing requirement is met as quickly as possible, and thereby minimizing its completion time.

We assume that each machine $i \in \mathcal{M}$ declares a policy, or a coordination mechanism, that determines how the speed $\nu_i$ is shared among the competing jobs. Given this policy, each job selects a single machine at every time $t$. Let $\theta(t) = (\theta_1(t), \theta_2(t), \ldots, \theta_n(t))$ denote the current configuration of jobs, where $\theta_i(t)$ denotes the machine a job $j$ chooses at time $t$. Furthermore, let $\nu(j, t)$ denote the speed assigned to a job $j$ at time $t$. In the next time instant $t + 1$, the jobs do a sequential best response in some order as follows (We assume that order the agents move is also determined by the mechanism designer): A job $j$ which is currently choosing the machine $\theta_i(t)$ switches to the machine $d \in \mathcal{M}, d \neq \theta_i(t)$, if and only if the speed it would receive on machine $d$ is greater than the speed it is receiving. This changes the speeds the jobs get on the machine $\theta_i(t)$ (since $j$ moved out) and the machine $d$. Therefore, at every time instant $t$, this defines a game on the jobs, and starting with the configuration at the previous step, the jobs do a sequential best response dynamics, where they sequentially change machines, till the system reaches a Nash equilibrium. This leads to a new configuration $\theta(t + 1)$, where each job is in a NE with respect to $\nu(j, t)$. We note that $\theta(t)$ need not change at each time instant; In fact, the configuration of jobs $\theta(t)$ changes only at time steps when either a job leaves the system or a new job arrives.

The cost of a job $j$ is the completion time $C_j$, which is defined as the earliest time instant at which the processing requirement of the job is met: $J_{t=0}^{C_j} \nu(j, t)dt = p_j$. The social cost is equal to the sum of players’ costs: $\sum_{j} C_j$. Our objective is to design a coordination mechanism that minimizes the inefficiency resulting from the selfish behavior of jobs. The benchmark we use to measure the inefficiency is the optimum offline solution to the underlying optimization problem, without involving any selfish behavior.

For the problem of minimizing the average completion time when all jobs arrive at the same time, we show that proportional sharing policy achieves a constant-factor approximation in equilibrium, for the related machines and restricted assignment models. Furthermore, for the problem of minimizing the total flow-time (or delay) when the jobs arrive online, we provide a constant-competitive algorithm for the related machines model with a constant speed augmentation. All the results are obtained by applying a dual-fitting technique.

---

†Department of Computer Science, Duke University . 308 Research Drive, Durham, NC 27708. kulkarni@cs.duke.edu. Supported by NSF Awards CCF-0745761, CCF-1008065, and CCF-1348696.
‡Google Research, New York.

Copyright is held by author/owner(s).