INCENTIVES AND GENERAL WELFARE FUNCTIONS IN THE OFF-LINE CLUSTER SCHEDULING PROBLEM

Orna Agmon

INCENTIVES AND GENERAL WELFARE FUNCTIONS IN THE OFF-LINE CLUSTER SCHEDULING PROBLEM

Research Thesis

Submitted in Partial Fulfillment of the Requirements
For the Degree of Master of Science
in Applied Mathematics

Orna Agmon

Submitted to the Senate of the Technion — Israel Institute of Technology

ADAR I 5763 HAIFA FEBRUARY 2003

The Research Thesis Was Done Under The Supervision of Dr. Rann Smorodinsky in In the Interdepartmental Program of Applied Mathematics

Acknowledgment

English Thanks

This Thesis is dedicated to the dedication of deduction.

Contents

Abstract

Several agents wish to execute one job each on a cluster- several computers which are a resource owned by an institution. The job's length is a secret known to the agent alone, and the agent's utility is the negation of the time in which she gets the output of the job. The institution would like to maximize a social welfare function over the agents' utilities by scheduling the jobs on the various computers. For example, the institution may wish to minimize the sum of times in which the agents get the output from their jobs, or to minimize the output time of the last job to come. The problem is that in order to act optimally, the institute must know the true lengths - an information it does not posses.

We construct mechanisms in which the agents prefer to tell the institution the real length of their jobs, while the institution maximizes a general social welfare function. Those mechanisms involve both monetary transfers before the execution, and implementation of job control tools during the execution. A job control tool is a certain algorithm for altering the initial allocation. Examples for such tools are postponing a part of the job for later, or lowering the share it gets of the CPU.

List of Symbols

Symbol	Meaning
$OLCS$	Off-Line Cluster Scheduling
CPU	Central Processing Unit
N	number of agents, also number of jobs
M	number of CPUs in the cluster
$\mathcal{M}$	a cluster of M CPUs
$\vec{\theta}$	vector of real job lengths
Θ	job type space
R_+	positive real numbers
$\vec{c}$	vector of processing power
A	allocation
$\mathcal{N}$	set of jobs of size N
$\mathcal{N}_m^A$	set of jobs which are executed on CPU m under allocation A
$X_n^A(t)$	work function of job n under allocation A at time t
$X^{A,m}(t)$	usage of CPU m under allocation A at time t
$\mathcal{A}$	the set of all allocations
$FUL(\vec{\theta})$	the set of allocations which fulfill $\vec{\theta}$
$\vec{T}$	output times vector
$\vec{B}^A$	beginning times vector under allocation A
$\vec{E}^A$	ending times vector under allocation A
Q	status
EFF	the set of all of the efficient allocations
$EFF(\vec{\theta})$	the efficient allocations which fulfill $\vec{\theta}$

Symbol	Meaning
$\vec{b}$	vector of broadcasted declarations
$Q(t)$	dynamic status at time t
$Q_{final}, Q(\infty)$	final status
$Q(0), Q_{static}$	static status
$\vec{L}$	vector of termination signals
Q_m	the status of CPU m
s'	a time in which the CPU is assured to be vacant, which is at least as large as the original POST parameter
$t_{firstsplit}$	the first time a split job on the CPU resumes execution
U_n	utility of agent n
V_n	agent n's valuation for the execution of her job
$\vec{P}(\vec{b})$	vector of price functions, which the agents pay the institution
g	social welfare function
$\mathcal{G}$	the set of regular social welfare functions
$\tilde{\mathcal{N}}_m^A$	the set of all the jobs on CPU m which are not a segment in time of value 1
$\tilde{\mathcal{N}}_{m,n}^A$	the set of jobs on CPU m which are active while job n is active
$i(k)$	an ordinal number of job k within set $\tilde{\mathcal{N}}_{m,n}^A$
$s_{i(k)}$	the amount of CPU time job $i(k)$ got while job n was active
$\tilde{\mathcal{N}}_{m,n}^A$	
s_{renice}	percentage in the RN operator
s_{post}	time in the $POST$ operator
F_n^A	job n's dependents
f_n^A	job n's dependency number
$o(\vec{x}, \vec{y})$	an allocation which is an outcome of a mechanism, when the declarations are $\vec{x}$ and the real types are $\vec{y}$
S	strategy
$\hat{\Theta}$	range of real lengths
$g_{\sum}$	the social welfare function which is the sum of utilities
W_n	utility of agent n in other environments from the allocation itself

Operator	Meaning
VCG	Vickrey Clarke Groves
E_n^{Q-}	time in which job n is supposed to end,
	according to the status prior to that time
Z	a mechanism
$\vec{T}(\vec{b},\vec{\theta})$	the vector of output times
	as a function of the declarations and the types
$\vec{U}(\vec{b},\vec{\theta})$	the vector of utilities
	as a function of the declarations and the types
S_{-n}	the strategies of all agents except agent n
b_{-n}	the declarations of all agents except agent n
θ_{-n}	the types of all agents except agent n
$d_n(\theta_n)$	the distribution function of the length of job n
$\vec{T}_\Sigma$	$\vec{T}$ in a system which optimizes $g_\sum$
$\vec{U}_\Sigma$	$\vec{U}$ in a system which optimizes $g_\sum$
$o_\sum$	o in a system which optimizes $g_\sum$
$\mathcal{N}_{-n}^A$	the jobs on the same CPU as job n, excluding job n
$Q^{k,-}$	the dynamic status
	at the time in which job k begins its execution
$D_n^A(\vec{b},\theta_{-n})$	the possible gained time
$EVCG$	Extended Vickrey Clarke Groves
H_{diff}	a limit for the downward lie, according to equation 5.13
H_{abs}	a limit for the real type, according to equation 5.14
$makespan^l$	the makespan function of order l, according to definition 6.1
ms^l	the makespan set of order l according to definition 6.2
$lexicalmax$	the lexical max function according to definition 6.2
$E^{m,A}$	the ending time of CPU m under allocation A
$\hat{\mathcal{N}}$	the subset of $\mathcal{N}$ which chose to stay for the second round

List of Operators

Operator	Meaning
$ES_n(r)$	operator "extended stay" for job n of time amount r
$RP_n(s,r)$	operator "reduced power" for job n to percentage s from time r
$GAP_n(r,s)$	operator "gap" for job n from time r until time s
$RN_n(s,r,\theta_n)$	operator "renice" for job n
	from time r to a percentage s until it performs a total work of θ_n
$CLOSE_m(r,s)$	operator "close gap" in computer m from time r until time s
$POST_n(r,s,\theta_n)$	operator "postpone" for job n from time r to time s
	and let it continue until it performs a total work of θ_n
$EARLY_n$	early release of job n

Chapter 1

Introduction

Let us examine an institution oriented at producing HPC (High Performance Computing) calculations. HPC calculations are characterized by long processing time, and no interactivity[1]. The people who send these computations are either workers of that institution, or outsiders who get the privilege of using the institution's resources. Those resources, which used to be composed of a mainframe[2] computer, are composed nowadays of a **cluster** of commodity personal computers (PCs): several PCs, with a common controlling center, which supply together the necessary computing power.

Each worker has one job to submit to the cluster, which may be composed of computers of varying strength. The cluster's qualities are known to the person submitting the jobs, but each job's length is the private information of the person who submits it. Though in the world of computer science, those workers would be referred to as "users", we shall use the term "agents", taken from game theory.

The agent's interest is that the job be finished as soon as possible: her utility from the completion of the job increases the sooner the job is finished. On the other hand, the institute has other interests, some of which may collide with those of the single agents. This institutional interest is often referred to as a **social welfare function**.

One example of an objective of the institution may be minimizing the sum of times in which all jobs end. Another would be minimizing the make-span[3] of the jobs.

[1] Some HPC calculations require a certain amount of interactivity, with long gaps between the interactive modes. One may look at them as several sequentially dependent calculations.

[2] Mainframe: a single, very strong, machine, which is usually shared among several uses at a time.

[3] Make-span- the time difference between the beginning of the first job, and the ending of the last. Also see

The classic problem that computer scientists and operations research scholars have treated is the computational difficulty of solving the optimization problem. For example, Du and Leung [13] prove that the off-line[4] scheduling problem, when the lengths of the jobs are known, is NP-hard. Others give procedures for reaching good results on specific cases. For example, Root [30] gives a procedure for off-line scheduling of multiple jobs on parallel machines with a common due date. Azizoglu and Kirca [6] give a heuristic algorithm for a similar problem, with different due dates.

However, this study looks at the off-line cluster scheduling problem from a different perspective, which results from the potential collision of interests among agents and between the agents and the institution.

1.1 The Scope of This Work

In this work, we assume the institution is capable of choosing an optimal schedule (a schedule which optimizes the social welfare function), given the exact lengths of the jobs. In subsection 7.7 we discuss the circumstances in which this assumption is valid.

We concentrate on designing a mechanism in which an optimal course of action for the agents is to reveal the length of their jobs, while the institution maximizes a general social welfare function. Thus we guarantee the maximization of the social welfare function. We design this mechanism using monetary transfers between the institution and the agents, and job control tools which change the allocation dynamically in a limited manner.

We limit the discussion to **off-line scheduling**, which means all the jobs[5] are available at time zero. Furthermore, we assume that the institution is able to control the execution of the jobs completely. The only actions on behalf of the agent are reporting the (not necessarily true) length of the job, and supplying the job.

The rest of this work is constructed as follows: In chapter 2 we refer to related work. In chapter 3 we describe the model, define the terms of allocation and job control tools, and the mechanism

definition 6.1 for $makespan^0$ and in Abdekhodaee and Wirth [1].

[4] **off-line scheduling** means the lengths of the jobs are all known at time 0. This comes as opposed to on-line scheduling, which means that jobs may be submitted while other jobs are being executed, as part of an on-going process. Different scheduling methods are discussed in section 2.

[5] for the current batch in which we deal. New jobs which are submitted, are performed in the next batch. We assume no interaction between those batches, i.e. jobs from a prior batch are not processed anymore after the processing of the current batch has begun.

derived from them. In chapter 4 we list several characteristics of desired mechanisms. In chapter 5 we implement a special social welfare function using certain combinations of job control tools and monetary transfers. In chapter 6 we extend the mechanism to implement a general social welfare function. In chapter 7 we discuss the quality of the proposed off-line cluster scheduling mechanisms, and we conclude in chapter 8.

Chapter 2

Related Work

2.1 Off-Line Scheduling games

2.1.1 Nisan and Ronen

Nisan and Ronen [28] present a problem where the agents are the CPU owners, whose interest is to do as little work as possible. The mechanism is required to allocate certain tasks to those agents. The CPUs are not related.

The authors propose various approaches for solving the problem of minimizing the makespan. They approximate the makespan welfare function using the minimal work welfare function, prove it is a tight bound and use randomness to improve that approximation.

They insert monetary payments after the execution, thus enabling the institution to verify how long indeed the jobs took on that specific CPU. Since this mechanism requires an exponential computation time, they provide an approximation for a sub-case of the problem: Nisan and Ronen also prove that for bounded types, their mechanism approximates the optimal value for the social welfare function, even if the institution only approximated the optimal allocation.

2.1.2 Van Ackere

Van Ackere [2] presents a game between an institution (which is also the scheduler) and any one agent. She deals with the problem of scheduling operation rooms in a hospital, when the real length

of operations is not known. The game she describes is held between the institution and any of the agents, who are the surgeons. The surgeon may arrive to the surgery later than scheduled, partly due to intentionally being late, and partly due to uncontrolled reasons. The surgeon's secret is the time in which he intended to arrive. The institution's utility decreases the later the surgeon is, relative to the time in which the operating room is ready. The institution tries to determine the proper time to schedule the operation.

In another work, Van Ackere [3] analyzes the strategies of a client and a dentist in two scheduling systems. In the first, the dentist performs the diagnosis and the treatment in one sitting. In the second, the dentist schedules two appointments, one for each of the tasks. While the diagnosis is of known duration, the treatment's length may vary, and it depends on the diagnosis. Van Ackere shows that there exist combinations of costs to the agents, a choice of a scheduling system, and a choice of arrival times, such that both agents would be better off in another combination (which is not in equilibrium).

2.1.3 Wellman, Walsh, Wurman and MackKie-Mason

In [38], Wellman, Walsh, Wurman and MackKie-Mason deal with an environment of one machine, on which time-units (time slots) need to be allocated to agents. Each agent has one job to process, for which several time units may be required before a deadline. An agent's utility from performing her job is her private information.

A solution (which is combined of an allocation and payments) is evaluated ex-post, according to the sum of utilities of the agents and the value of time slots which were not allocated. The authors present a decentralized market-inspired protocol for reaching solutions in this environment.

2.2 Queuing Games

2.2.1 Naor

Naor [27] was the first to compare individual optimality to social welfare in a queue based on the FCFS discipline, and to use payments, in order to implement a social welfare function.

2.2.2 Altman and Shimkin

Altman and Shimkin [4] analyze, from the queuing theory point of view, an environment where the agents may choose between sharing a "mainframe", a strong, common, computational resource. Every agent has access to an alternative, exclusive, slower CPU. If the load on the mainframe is higher than a certain threshold, i.e. the CPU share the agent gets is too low, the agent will prefer the private option.

2.2.3 Van Mieghem

Van Mieghem [24] proposes a "generalized c μ rule " scheduler for one server. The scheduling policy approximates optimality of the social welfare function of the sum of weighted utilities $-\sum W_k T_k$, when the agent's cost function from the output time is a non-decreasing, convex function and the server is close to its full capacity. These results are valid for a vast range of input distributions.

2.3 Variations On The Cluster Scheduling Problem

2.3.1 Migration

Migration is a way to correct scheduling mistakes. When the institution sees that a certain job is running on a non-optimal CPU, it can make it migrate (move it) to another CPU in the cluster. This solution requires the jobs to be preemptive[1], and the system to support migration. Migration itself has a performance cost due to the time it takes to move the job between CPUs, and due to ongoing calculations to determine whether a migration should happen. Hence, a migration is worth performing only if the current situation is non-optimal beyond a certain level. MOSIX [8, 7], a Linux [36] based "single system image"[2] cluster operating system is an example for a migration-based system.

[1] A **preemptive** job is a job whose operation can be suspended and then resumed.

[2] A **single system image** is a concept which means making the cluster appear like one strong machine to the users.

2.3.2 Off-Line Scheduling

Tanenbaum [34] describes the scheduling algorithm of "Shortest Time First" (STF), which yields an optimal schedule for a single CPU, when the institution wishes to minimize the sum of completion times. As indicated there, the problem the institution faces is that it has no knowledge of the jobs' lengths, so it cannot schedule them to optimize the social welfare function.

2.3.3 On-Line Scheduling

When the time and number of arriving jobs is unknown (except for maybe a probability distribution), **on-line** scheduling is called for. The institution must schedule the jobs as they are submitted, possibly making decisions which are based on the statistical prediction of arrivals. These systems may contain a queue of jobs which arrived, but has not begun yet. Often these systems are referred to as **queuing systems**. Usually, the algorithms used in these systems are heuristics, which do not yield an optimal schedule, but a certain approximation. For example, Condor [22], a high throughput computing system, and Jobd [20], a utility for job management on the GNU/Linux [35] operating system.

2.3.4 Backfilling

Backfilling, a term coined by Lifka [21] for the EASY scheduler for the IBM SP1 computer, is a method for scheduling a cluster, when the jobs require the use of several CPUs at the same time. If served according to FCFS (First Come First Served), jobs which require a large number of CPUs cause cluster idleness (while waiting for enough CPUs to be free). On the other hand, letting jobs which require a smaller amount of CPUs to leapfrog the vastly requiring job may lead to starvation. Feitelson and Mu'alem Weil [15] compare variants of backfilling. Conservative backfilling: permits such leapfrogs just for jobs which do not cause any delay to other jobs. This is done by asking the agents for the expected length of their jobs. The agents have an incentive to give a short estimation, since a shorter job is more likely to leapfrog, and will be scheduled sooner. On the other hand, using job control tools, the job is killed (its processing is stopped) if it exceeds the declared length, thus giving the agents incentive to declare a length which is at least as long as the real length. This system also "compresses" the allocation, in order to close gaps. In this mechanism the agent's utility

drops sharply when the declaration is shorter than the real length of the job. Comparing the final outcome to the FCFS priority, it is "predictable" (or "just"): for a truth telling agent, every job's output time may only improve.

In EASY backfilling, jobs may leapfrog if they do not cause any delay to the first job in the queue, thus losing the "predictability" property.

An important part of this work deals with the quality of the estimates. Even though the agents knew the system had only one sided safety margins, 99.7% of them failed to declare a length longer than the real length, and their jobs were killed before termination time.[3]

2.3.5 On-Line Schedulers Which Involve Monetary Transfers

These schedulers usually use market mechanisms to set the price, assuming the institutions owning the cluster aim at making a profit.

REXEC

REXEC [14] is a scheduler which enables the agent to specify the maximal cost per minute that she is willing to pay, and allocates her job according to that requirement.

Nimrod-G

Nimrod/G [9] supplies an economic environment for negotiations, auctions and other economical mechanisms in order to match jobs with CPUs.

Libra

Libra [31] is an economy based on-line cluster scheduler. It requires the agent to specify three numbers when submitting the job:

1. E The length of the job, in terms of the estimated time it would take to execute the job on a stand alone CPU
2. D A deadline, which is a time.

[3]The authors claim that it is likely that some of those jobs had used checkpoints, so that not all the work was lost (having checkpoints means the agent is capable of continuing her job from a point close to where the job was killed.

3. B a budget- an amount of money the agent is willing to pay.

The institution tries to deliver the output by the deadline, while charging the agent no more than the budget. Upon submitting, the institution estimates the minimal cost as

$$cost = \alpha E + \beta \frac{E}{D},$$

where α, β are coefficients. If the estimated cost is higher than the budget, the institution refuses to perform the job, and the agent may submit it again with relaxed conditions.

The authors assume the agents tell the truth regarding the length of their jobs, while there might be a trade-off between the deadline and the budget. Knowing the other agents' declarations, and the algorithm the scheduler uses, an agent might deliberately declare a wrong value as a job length, in order to manipulate the allocation.

2.3.6 Distributed Decisions

The match making between the job and the CPU which will execute it can be taken in a central decision-maker, or in a distributed manner. The distributed manner enables the scaling of the cluster. Stankovic and Sidhu [32] present an algorithm for match-making, in which every job sees just a part of the CPUs, and vise versa. This algorithm requires full disclosure of information on both sides, in the form of contracts: every job must state its needs, and every CPU must state its capabilities. MOSIX and REXEC also take the distributed decision approach, but Condor uses the centralized approach.

Chapter 3

The model

In this chapter we describe the mathematical model for the allocation of jobs of certain lengths to a cluster of certain CPU speeds. We then describe the job control tools, which are operators used after the execution of the jobs had began, in order to make specific alterations to the schedule. We describe the conflict between the agent's utility and the institution's utility, and formulate the mechanism, which is the framework for solving that conflict.

3.1 The Primitives

N agents, each with a single job which needs processing, submit their jobs to a cluster of M CPUs[1] at time zero. The length of each job is only known to the agent. Each agent also declares the length of the job: the amount of work required to process it. This declaration is assumed to serve the interest of the agent, and consequently, is not necessarily the true length.[2] An Off-Line Cluster Scheduling Algorithm assigns each job to a given computer in the cluster based on the information provided by the submitting agents. Ideally the OLCS algorithm would have the actual true parameters of each job (i.e., its length) and would optimize cluster assignment based on that.

[1] When talking about a cluster, it is customary to deal with the term "node", the basic computing entity in the cluster. One computer may be associated with more than one node, especially if it has more than one CPU in it (an SMP machine). In this work we assume every CPU is matched with exactly one node. We neglect the deficiency in performance caused by sharing devices, e.g. I/O and network devices, so in other words we may say that we assume every computer has just one CPU.

[2] In contrary to a large part of the mechanism design literature, for example Mas-Colell et al. [23], the agent's type (her secret) is not in the utility function space.

In our setup the OLCS algorithm asks for the information and agents may choose to truly reveal it or lie about it.

We shall denote by $\vec{\theta} = (\theta_1, \ldots, \theta_N) \in \Theta^N$, $\Theta = \mathbb{R}_+$, a profile of jobs, where N is the number of agents and for each $1 \leq n \leq N$, θ_n is its true length. We also denote by $\vec{c} = (c_1, \ldots, c_M) \in \mathbb{R}_+^M$ a cluster of M CPUs, where the m^{th} CPU has processing power c_m. We assume the machines are **related**: machines which can be characterized by a single number c_m, such that the time to complete work θ on CPU m is exactly $\frac{\theta}{c_m}$.[3] While the actual units used along this work are not important, there is a connection between the length and power units: the CPU power is measured in length units divided by the time units used.

Another assumption we take is that the jobs are **CPU-hungry** during their entire execution: at any point in time, if the CPU is busy executing one job, it cannot contribute to the execution of another job without delaying the first job[4].

$\vec{c}$ is common knowledge, while the job's real length, θ_n, is private information of agent n.

3.1.1 Allocations

The first task done by an OLCS algorithm is to define which job runs on which computer and in which order. We refer to this as an *allocation* and define it as follows:

Definition 3.1 *An* **allocation** A *of a set of jobs* $\mathcal{N}$ *is composed of:*

1. **A partition** $\{\mathcal{N}_m^A\}_{m \in \mathcal{M}}$ *of* $\mathcal{N}$*, namely :*$\forall m \in \mathcal{M}$, $\mathcal{N}_m^A \subset \mathcal{N}$ *is a subset of* $\mathcal{N}$ *s.t.*

$$\cup_{m \in \mathcal{M}} \mathcal{N}_m^A = \mathcal{N}$$
$$\forall m \neq k \quad \mathcal{N}_m^A \cap \mathcal{N}_k^A = \emptyset \tag{3.1}$$

2. **Work functions***:* $\forall m \in \mathcal{M}, \forall n \in \mathcal{N}_m^A$, $X_n^A(t) : \mathbb{R}_+ \mapsto [0,1]$ *denotes the percentage of CPU* m *which is devoted to job* n *at time* t*, and satisfies* $\sup\{t : X_n^A(t) > 0\} < \infty$*, as well as* $\forall m \in \mathcal{M}$

$$X^{A,m}(t) := \sum_{n \in \mathcal{N}_m^A} X_n^A(t) \leq 1$$

The work functions are continuous to the right.

[3] We ignore architectural differences, memory-size differences etc.

[4] In real life, this may not be the case: while one job is waiting for a resource, such as an I/O device or some network input, another job may be processed without disturbing the first one. On the other hand, when processing more than one job at the same time, both jobs suffer a penalty of context switching. Hence, some social welfare function may improve by running jobs in parallel.

We refer to the function $X^{A,m}$ as the **usage** *of computer m under allocation A.*

The property defined by equation (3.1) is the **no migration** property, to which we limit ourselves in this work.[5] The set of all allocations is denoted by $\mathcal{A}$.

For a given profile of jobs, $\vec{\theta}$, let us consider the subset of allocations that devote the exact resources needed for computing:

Definition 3.2 *An allocation A* **fulfills** *a vector of job lengths $\vec{\theta}$ if*

$$\forall m \in \mathcal{M} \;\; \forall n \in \mathcal{N}_m^A \;\; c_m \int_{t=0}^{\infty} X_n^A(t)dt = \theta_n.$$

The set of allocations which fulfill $\vec{\theta}$ is denoted by $FUL(\vec{\theta})$.

Definition 3.3 *For every allocation A and job n, we denote the* **beginning time** *of job n by $B_n^A = inf\{t : X_n^A(t) > 0\}$ and the* **ending time** *of job n by $E_n^A = \sup\{t : X_n^A(t) > 0\}$. By T_n we denote the* **output time**, *the time in which the output of job n is given to the agent.*

Definition 3.4 *We say that a vector of output times, $\vec{T} := \{T_n\}_{n=1}^N$, is* **supported** *by allocation A, if $T_n \geq E_n^A$ for all n.*

Definition 3.5 *We refer to the tuple $Q = (A, \vec{T})$, where $\vec{T}$ is supported by A, as the* **status** *of the system. We abuse notation and define B^Q as B^A, and E^Q as E^A.*

3.1.2 Efficiency

Definition 3.6 *An allocation A is* **efficient** *if*

1. *$X^{A,m} \in \{0,1\}$ and is a non-increasing function, $\forall m \in \mathcal{M}$. In other words: there are no gaps in the usage of a CPU. The CPU is never partially used. If it is used at all, then it is used at time 0.*

2. *$X_n^A(t) \in \{0,1\}$ and is an interval in $\mathbb{R}_+$,$\forall n \in \mathcal{N}$. In other words, there are no gaps in the execution of a job, nor is it executed with just a part of the CPU at any time.*

[5] In SMP machines, the operating system handles migration between CPUs in the same computer. As we mentioned in footnote 1 in chapter 3, we assume in this work that every computer has only one CPU.

Let EFF be the set of all efficient allocations, and let

$$EFF(\vec{\theta}) := FUL(\vec{\theta}) \bigcap EFF$$

The notion of efficiency we defined here differs from the notion of Work-Function-Pareto efficiency:

Definition 3.7 $\forall \vec{\theta} \in \Theta^N$, *allocation* $A \in FUL(\vec{\theta})$ *is* **Work-Function-Pareto-efficient** *if* $\nexists A' \in FUL(\vec{x})$ *s.t.* $\exists n \in \mathcal{N}$ *s.t.:*

$$\begin{aligned} \forall l \neq n \quad , \quad \forall t \geq 0 \quad X_l^{A'}(t) &= X_l^A(t) \\ E_n^A &> E_n^{A'}. \end{aligned}$$

In other words, allocation A is Work-Function-Pareto efficient if there is no change that can be done to it, which involves just one work function, such that the ending time of that job is improved.

Example 3.1.1 *A Work-Function-Pareto-efficient allocation is not necessarily efficient.*

Proof: For example, let us take $\mathcal{M} = \{1, 0.1\}$, $\vec{\theta} = \{2, 1\}$. Allocation A_{pe} is Work-Function-Pareto-efficient, but it is not efficient.

$$\mathcal{N}_1^{A_{pe}} = \{1, 2\}$$

$$X_1^{A_{pe}}(t) = \begin{cases} 1 & 0 \leq t < 1 \\ 0 & 1 \leq t < 2 \\ 1 & 2 \leq t < 3 \\ 0 & 3 \leq t \end{cases}$$

$$X_2^{A_{pe}}(t) = \begin{cases} 0 & 0 \leq t < 1 \\ 1 & 1 \leq t < 2 \\ 0 & 2 \leq t \end{cases}$$

The gaps in the execution of job number 1 make the allocation inefficient, but there is no alteration of $X_1^{A_pe}$ alone such that the ending time of job 1 will be sooner than 3. ■

Example 3.1.2 *An efficient allocation is not necessarily Work-Function-Pareto-efficient.*

Proof: Let us examine allocation A_{ef} on the same cluster, with the same set of jobs as A_{pe}.

$$\mathcal{N}_2^{A_{pe}} = \{1, 2\}$$

$$X_1^{A_{ef}}(t) = \begin{cases} 0 & 0 \leq t < 10 \\ 1 & 10 \leq t < 30 \\ 0 & 30 \leq t \end{cases}$$

$$X_2^{A_{ef}}(t) = \begin{cases} 1 & 0 \leq t < 10 \\ 0 & 10 \leq t \end{cases}$$

This allocation, though efficient, is not Work-Function-Pareto-efficient. Both agents would be better off executing their jobs on CPU number 1, even if they move there without changing any other work function but their own. ■

Definition 3.8 *We say that efficient allocations* A_1, A_2 **share the same schedule** *if :*

$$\begin{aligned} \forall n \in \mathcal{N} \quad & n \in \mathcal{N}_m^{A_1} \quad \Leftrightarrow \quad n \in \mathcal{N}_m^{A_2} \\ \forall m \in \mathcal{M}, \forall k, l \in \mathcal{N}_m^{A_1} \quad & B_k^{A_1} > B_l^{A_1} \quad \Leftrightarrow \quad B_k^{A_2} > B_l^{A_2} \\ \forall m \in \mathcal{M}, \forall k, l \in \mathcal{N}_m^{A_1} \quad & E_k^{A_1} > E_l^{A_1} \quad \Leftrightarrow \quad E_k^{A_2} > E_l^{A_2}. \end{aligned}$$

In other words, efficient allocations share the same schedule if the matching of jobs to computers is the same, and the order in which jobs are executed within the computer is also the same.

3.2 Job Control Tools

Job control tools are used in order to alter the allocation after the cluster has begun executing the jobs according to it.

The execution stage comes after a set of jobs is assigned to computers via an allocation. During an execution some of the plans could contradict reality, as a consequence of agents not reporting

truthfully. For example, if a job ends prematurely, then the real ending time could be shorter than planned. In the period of time between the real ending time and the planned one, the CPU would be used by the job less than planned.

Consequently, for a set of a real allocation and real jobs, we have the following definitions, which enable changing the allocation dynamically. We refer to the initial status as a **static** status, and to the status after the changes as a **dynamic** status. We denote the dynamic status at time t by $Q(t)$, and the static status as $Q(0)$ or Q_{static}. By $Q(\infty)$ or Q_{final} we denote the **final** status.

Recall that in our set-up, the initial scheduling is done based on the reported job lengths, which may be quite different from the true lengths. Consequently, job execution may end sooner than planned. More formally:

Definition 3.9 *We denote the* **termination time** *of job n, the time in which job n was terminated, by $L_n \in \mathbb{R}_+$. This time is only known to the institution after the job executes work θ_n.*[6] *The termination time of job n fulfills:*

$$c_m \int_{t=0}^{L_n} X_n^{Q(t)} dt = \theta_n$$

3.2.1 Intuition For Job Control Tools

The term "job control" is used by system administrators to describe various ways by which one can interfere with the CPU allocation during the execution of processes, in particular, ways to suspend the execution and resume it at a later stage[7].

Note that not all job control tools are necessarily available on every system, as indicated in Stevens [33].

We focus on certain job control tools, which present a mathematical model of existing tools, some of which are operated manually, and some automatically.

1. Renice[8]: The job gets only a share of the CPU, but for enough time to complete its execution.

[6] On the other hand, this information cannot be hidden from the institution.

[7] Job Control is used by various Unix shells, e.g. bash [11]. Job Control is also used by Jobd [20] as a manner of punishment: When a job is submitted to Jobd, the user supplies certain demands of memory and CPU. If the job uses less, Jobd tracks that down, and lowers the resources allocated to it. If the job uses more than declared, the job is punished by a "renice" operation- the amount of resources it gets is limited.

[8] Based on the *renice* command on Unix systems.

2. Postpone[9] : the job stops receiving any computation power for some time, then it gets enough share of the CPU for enough time to complete its processing.

3. Close-Gap[10]: A **gap** is a period of time in which the CPU stands idle[11], though there are jobs or parts of jobs awaiting that CPU. This tool closes that gap by performing the tasks which await the same CPU.

4. Early Release[12] : The output of the job is given to the agent prior to the time agreed upon, as soon as the processing of the job is done. We assume every system can use the Early Release tool, as it is a matter of policy rather than availability.

Some of those tools are actually a combination of more basic tools, which are not useful on their own. The tool *Renice* is a combination of the two following basic tools:

- Extended-Stay: Though the job was supposed to be stopped at a certain time, it gets an extra amount of time.
- Reduced power: The job's share of the CPU changes. However, the time in which the job's processing is stopped is not changed.

The tool Postpone is a combination of the basic tool Extended-Stay and the basic tool Gap:

- Gap[13]: The job does not get any share of the CPU for a certain amount of time. At a later time, the job receives a positive share of the CPU again.

3.2.2 Formal Definitions for Job Control Tools

We turn to formalize the aforementioned job control tools in the form of operators over allocations. As we exclude job migration, those operators do not change the partition: the sets $\mathcal{N}_m^A$ are set before the execution stage, and do not change. As we later prove, it is optimal not to perform any changes to the static allocation, hence the lack of migration does not impose a major constraint.

[9] Postpone, for example, is not possible if preemptions are not allowed.

[10] Resembles regular behavior on Unix Systems, when the commands are separated by ";": The next command begins its execution once the first one has ended.

[11] Let us distinguish between **full gaps**, gaps in which the CPU stands totally idle (its usage being 0), and **partial gaps**, when the the usage drops to a positive value. In this work we allow closing of full gaps only, to avoid complex definitions.

[12] Typical of large clusters, such as the ASCI initiative [29, 39], where the agents do not have direct access to the machine. This, unlike the former two, is not a common Unix tool.

[13] Resembles the combination of *suspend* and (after a gap of time) *fg* on Unix systems.

We begin with several preliminary definitions:

Definition 3.10 *$\forall m \in \mathcal{M}, \forall Q \in (\mathcal{A}, \mathbb{R}_+^N)$, let*

$$Q_m := (\{X_k : k \in \mathcal{N}_m\}, \{T_k : k \in \mathcal{N}\}).$$

Then Q is a Cartesian product:

$$Q = \times_{m \in \mathcal{M}} Q_m.$$

We refer to Q_m as the **status of CPU m**.

Note that once the static statuses are set, changes done to the status of one CPU, by performing an operator on it, do not affect the status of any other CPU. In other words, let us relate to operators of two kinds:

- An operator on CPU m: OP_m on a status Q, such that OP_m makes alterations to X_k^A, $\forall k \in \mathcal{N}_m^A$.
- An operator on job n: OP_n on a status Q, such that OP_n makes alterations to X_n^A, where $n \in \mathcal{N}_m^A$, and it may affect also other work functions X_k^A where $k \in \mathcal{N}_m^A$

Then,

$$(OP_m(Q))_k = Q_k \; \forall k \neq m$$
$$(OP_n(Q))_k = Q_k \;\; \forall n \in \mathcal{N}_m^A \; \forall k \neq m.$$

Definition 3.11 *$\forall r \in \mathbb{R}_+, \forall n \in \mathcal{N}$, we define the job control operator $ES_n(r)$, referred to as* **extended-stay for job** *n* **of time amount** *r, on a status Q as follows, where $n \in \mathcal{N}_k^A$ and s stands for $\liminf_{t \to E_n^A}(X_n^A(t))$[14]:*

$$X_n^{ES_n(r)(Q)}(t) = \begin{cases} s & r + E_n^A > t \geq E_n^A \\ X_n^A(t) & otherwise \end{cases}$$

[14]Defining s as $\liminf_{t \to E_n^A}(X_n^A(t))$ instead of $X_n^A(E_n^A)$ is a solution for the technical difficulty, resulting from assuming that the work functions are right-continuous only, which means that $X_n^A(E_n^A) = 0$, while $X_n^A(B_n^A) > 0$.

for $l \neq n$:

$$X_l^{ES_n(r)(Q)}(t) = \begin{cases} X_l^A(t) & l \notin \mathcal{N}_k^A \\ X_l^A(t) & l \in \mathcal{N}_k^A \quad and\ 0 \leq t < E_n^A \\ 0 & l \in \mathcal{N}_k^A \quad and\ E_n^A \leq t < E_n^A + r \\ X_l^A(t-r) & l \in \mathcal{N}_k^A \quad and\ r + E_n^A \leq t \end{cases}$$

$$T_l^{ES_n(r)(Q)} \quad = \quad \max(E_l^{ES_n(r)(Q)}, T_l^A)$$

Note that the extended jobs are not made to run in parallel to other jobs, even if they do not use all of the CPU.

Definition 3.12 *$\forall s, r \in \mathbb{R}_+, \forall n \in \mathcal{N}$, we define the job control operator $RP_n(s,r)$, referred to as* **reduced power for job** *n* **to percentage** *s* **from time** *r, on a status Q as follows:*

$$X_l^{RP_n(s,r)(Q)}(t) = \begin{cases} sX_n^A(t) & if\ l = n\ and\ t \geq r \\ X_l^A(t) & otherwise \end{cases}$$

$$\vec{T}^{RP_n(s,r)(Q)} \quad = \quad \vec{T}^Q.$$

Note that if $s < 1$, then $RP_n(s,r)$ does not preserve the efficiency. More over: if A fulfills $\vec{\theta}$ and $s < 1$, then $RP_n(s,r)(Q)$ will not fulfill $\vec{\theta}$. Though we make a general definition here, we do not use RP on its own in this work: RP is used only to define the operator RN, and in this context we make sure that the final allocation fulfills the vector of real lengths $\vec{\theta}$.

As part of the $POST$ operator, we define the GAP operator. GAP transfers a part of the work that needs to be done for a job to another time in the future, on the same CPU. In order to make sure that the allocation formed by this operator is well-defined, the rest of the work is transferred to some future time s', at which the CPU is surely vacant. The GAP operator does not preserve efficiency, either.

Definition 3.13 *$\forall s, r \in \mathbb{R}_+, \forall n \in \mathcal{N}$, we define an operation $GAP_n(r,s)$,* **gap in job** *n* **from time** *r* **until time** *s on a status Q, in which $X_n(t) = 1\ \ \forall r < t < E_n^A$, as follows:*

$$X_n^{GAP_n(r,s)(Q)}(t) = \begin{cases} X_n^A(t) & 0 \leq t < r \\ 0 & r \leq t < s' \\ X_n^A(t - (s' - r)) & s' \leq t \end{cases}$$

where $n \in \mathcal{N}_k^A$, $s' = \max(s, \sup\{t : X_k^A(t) \neq 0\})$.

For $l \neq n$:

$$\begin{aligned} X_l^{GAP_n(r,s)(Q)}(t) &= X_l^A(t) \\ T_l^{GAP_n(r)(Q)} &= \max(E_l^{GAP_n(r)(Q)}, T_l^Q) \end{aligned}$$

Note that the definition of GAP, and in particular the use of s', does not permit creating an allocation in which two jobs run in parallel on the same CPU.

Now we turn to define the job control tools we will actually refer to later on:

Definition 3.14 $\forall s, r \in \mathbb{R}_+$, $\forall n \in \mathcal{N}$, $\forall \theta_n \in \Theta$, *we define the job control operator* $RN_n(s, r, \theta_n)$ *referred to as* **renice for job** n **from time** r **to a percentage** s **until it performs a total work of** θ_n *on a status* Q *as follows:*

$$RN_n(s, r, \theta_n)(Q) = RP_n(0, L_n) \circ ES_n(\infty) \circ RP_n(s, r)(Q)$$

where L_n, *the termination time, is implicitly defined by*

$$c_m \int_{t=0}^{L_n} X_n^{ES_n(\infty) \circ RP_n(s,r)(Q)} dt = \theta_n$$

where $n \in \mathcal{N}_m^A$.

In order to define the $CLOSE$ operator, we need to define split jobs. A job is split if its work function is not an interval in time. More formally:

Definition 3.15 *Job* $n \in \mathcal{N}_m^A$ *is* **split** *if*

$$\exists t' \; s.t. \;\; B_n^A < t' < E_n^A \;\; and \; X_n^A(t') = 0$$

We denote the set of split jobs in $\mathcal{N}_m^A$ *by* $SPLIT\left(\mathcal{N}_m^A\right)$.

Definition 3.16 $\forall s, r \in \mathbb{R}_+, \forall n \in \mathcal{N}$, *we define an operation* $CLOSE_m(r, s)$, **close gap in computer** m **from time** r **until time** s *on a status* Q, *in which* $X^{m,A}(t) = 0 \; \forall r \leq t < s$, *as follows:*

$$\vec{T}^{CLOSE_m(r,s)(Q)} = \vec{T}^Q$$

$$X_n^{CLOSE_m(r,s)(Q)}(t) = \begin{cases} X_n^A(t+s-r) & r \le t < t_{firstsplit} \text{ and } n \in \mathcal{N}_m^A \\ X_n^A(t) & otherwise \end{cases}$$

where $t_{firstsplit}$ *stands for*

$$\inf\{t : \exists k \in SPLIT\left(\mathcal{N}_m^A\right), \exists B_k^A < t' < t \text{ s.t. } X_k^A(t) > 0 \text{ and } X_k^A(t') < 1\}$$

In other words, $t_{firstsplit}$ *is the first time in which a split job on CPU* m *resumes its execution.*

The limitation $t < t_{firstsplit}$ enables activating $CLOSE$ after GAP, or even several GAP operations, without undoing those GAP operations.

Definition 3.17 $\forall s, r \in \mathbb{R}_+, \forall n \in \mathcal{N}$, *we define an operation* $POST_n(r, s, \theta_n)$, **postpone job** n **from time** r **to time** s **and let it continue until it performs a total work of** θ_n *on a status* Q *as follows:*

$$POST_n(r, s, \theta_n)(Q) = RP_n(0, L_n) \circ CLOSE_m(r, s) \circ ES_n(\infty) \circ GAP_n(r, s)(Q)$$

where $n \in \mathcal{N}_m^A$, *and* L_n *is implicitly defined by*

$$c_m \int_{t=0}^{L_n} X_n^{CLOSE_m(r,s) \circ ES_n(L_n) \circ GAP_n(r,s)(Q)} dt = \theta_n.$$

Note that the $POST$ and RN operators, as defined, cannot be used simultaneously on one system, for if they are, then it might occur that at time s' in which the CPU was supposed to be idle, a job which was reniced is still running.

Definition 3.18 $\forall n \in \mathcal{N}$, *we define an operation* $EARLY_n$, **early release of job n** *on a status* Q, *as follows:*

$$T_l^{EARLY_n(Q)} = \begin{cases} T_l^Q & l \neq n \\ min(E_n^A, L_n) & l = n. \end{cases}$$

$$\forall t \ge 0, \ \ \forall l \in \mathcal{N} \ \ X_l^{EARLY_n(Q)}(t) \ \ = \ \ X_l^A(t)$$

Note that $EARLY$ is only well-defined at time L_n, and only if $L_n \le E_n^A$. We constrain the use of $EARLY$ to those cases only.

3.3 Utilities

We assume in our model that all agents behave in order to maximize some utility function which is known to all, and is a linear combination of the time it took them to receive the results of their job with some monetary transfer. On the other hand, we assume that the institution also behaves to maximize its utility, which is derived from the final status. The institution ignores the monetary outcome.

3.3.1 Agents' Utilities

Assume each agent n receives her job's results at a positive time T_n. Also assume that every agent is required to pay a price P_n to the institution for the execution of her job ($P_n \in \mathbb{R}$ does not have to be positive). Assume the agent has a valuation V_n for a performed job. Then the agent's utility U_n is

$$U_n = V_n - T_n - P_n. \tag{3.2}$$

As we currently ignore individual rationality[15] considerations, we may assume w.l.o.g. that $V_n = 0$, $\forall n \in \mathcal{N}$.

3.3.2 Social Welfare

Definition 3.19 *A* **social welfare function** *is a function* $g : \mathbb{R}_+^N \mapsto \mathbb{R}$. *The set of all social welfare functions is denoted by* $\mathcal{G}$.

Definition 3.20 $\forall n \in \mathcal{N}$, *for any vector* $\vec{z} \in \mathbb{R}^N$, *let* $\vec{z}$ *be decomposed as* $\vec{z} = (z_n, z_{-n})$, *where* $z_{-n} \in \mathbb{R}^{N-1}$.

Definition 3.21 *A social welfare function is* **regular** *if* $\forall z_{-n} \in \mathbb{R}_+^{N-1} \;\; \forall x, y \in \mathbb{R}_+$

$$x > y \Rightarrow g(z_{-n}, x) < g(z_{-n}, y).$$

For example, the social welfare function

$$g = -\sum_{k \in \mathcal{N}} T_k$$

[15] An individually rational agent will only participate if her utility from participating exceeds her utility from not participating, which is considered 0. We currently assume all agents must participate. In section 7.5 we discuss individually rational agents, who may choose not to participate.

is a regular social welfare function. We denote that particular social welfare function by $g_{\sum}$.

Lemma 3.22 *$g_{\sum}$ is a regular social welfare function.*

Proof: For an arbitrary $z_{-n} \in \Theta^{N-1} \subseteq \mathbb{R}_+^{N-1}$, and $x, y \in \Theta \subseteq \mathbb{R}_+$ such that $x > y$, we have

$$\begin{aligned} g_{\sum}(z_{-n}, x) = -x - \sum_{p=1}^{N-1} z_p &< \\ -y - \sum_{p=1}^{N-1} z_p &= g_{\sum}(z_{-n}, y). \end{aligned}$$

Therefore, $g_{\sum}$ is regular. ■

The function $-g_{\sum}$ is an example for a non-regular social welfare function.

Lemma 3.23 *If g is a regular social welfare function and*

$$A \in argmax_{a \in FUL(\vec{\theta})} g(\vec{E}^a)$$

then $X^{A,m}(t) \in \{0, 1\}$ and is a monotonic decreasing function (in the weak sense) $\forall t \geq 0$, $\forall m \in \mathcal{M}$.

Proof: Assume in contradiction that $X^{A,m}$ is not non-decreasing. Then $\exists \epsilon, \delta > 0$, $t_2 > t_1 + \epsilon$ such that $1 \geq X^{A,m}(t_2) - \delta > X^{A,m}(t_1)$.

Alternatively, assume in contradiction $\exists t_1$ *s.t.* $0 < X^{A,m}(t) < 1$. Since $X^{A,m}$ is continuous to the right, $\exists \delta > 0, \epsilon > 0$, such that $t_2 > t_1 + \epsilon$, and $\forall t_1 \leq t < t_2$ $X^{A,m}(t) < 1 - \delta$.

Then, in either of the cases, the CPU can perform at least an extra work of $\delta(t_2 - t_1)c_m$. Let $E^{m,A} := \max_{k \in \mathcal{N}_m^A} E_k^A$. Let n be a job such that $n \in \mathcal{N}_m^A$ and $E_n^A = E^{m,A}$.

Let us define allocation A_2:

$$\forall j \in \mathcal{M} \quad \mathcal{N}_j^{A_2} = \mathcal{N}_j^A$$

$$X_l^{A_2}(t) = \begin{cases} X_l^A(t) + \delta & t_1 \leq t < t_2 \text{ and } l = n \\ X_l^A(t) & \text{otherwise} \end{cases}$$

In this allocation we have added more work to job n. Let $\vec{\theta}$ denote the vector of lengths A fulfills. Then exists a unique $t_3 < E^{m,A}$ such that

$$\begin{aligned} c_m \int_0^{t_3} X_n^{A_2}(t)dt &= \theta_n \\ c_m \int_{t_3}^{E_n^A} X_n^{A_2}(t)dt &= \delta(t_2 - t_1)c_m. \end{aligned}$$

Let us define allocation A_3:

$$\forall j \in \mathcal{M} \quad \mathcal{N}_j^{A_3} = \mathcal{N}_j^A$$

$$X_l^{A_3}(t) = \begin{cases} X_l^A(t) + \delta & t_1 \leq t < t_3 \quad \text{and } l = n \\ 0 & t_3 \leq t \qquad \text{and } l = n \\ X_l^A(t) & \text{otherwise} \end{cases}$$

Allocation A_3 fulfills $\vec{\theta}$, since we built it so for agent n, and we did not change the other work functions:

$$\begin{aligned} \forall l \neq n \;\; X_l^A &= X_l^{A_3} \\ \forall l \neq n \;\; E_l^A &= E_l^{A_3} \\ E_n^A &> E_n^{A_3} \end{aligned}$$

Then $g(\vec{E}^{A_3}) > g(\vec{E}^A)$, and $A \notin argmax_{a \in FUL(\vec{\theta})} g(\vec{E}^a)$. ■

Corollary 3.3.1 *If allocation A is Work-Function-Pareto-efficient, then $X^{A,m} \in \{0,1\}$ and is a monotonic decreasing function (in the weak sense).*

Proof: If it does not hold for allocation A that $X^{A,m} \in \{0,1\}$ and is a monotonic decreasing function (in the weak sense), then build allocation A_3 as indicated in the proof of lemma 3.23. Under allocation A_3 one of the agents is better off, while the others' work functions are not changed, hence A cannot be Work-Function-Pareto efficient. ■

Lemma 3.24 *If g is a regular social welfare function and $A \in argmax_{a \in FUL(\vec{\theta})} g(\vec{E}^a)$ then $X_n^A(t) \in \{0,1\}$ and is a segment in time, $\forall n \in \mathcal{N}$, $\forall t \geq 0$.*

Proof: Let us assume by contradiction that on CPU m there is at least one job whose work function is not a segment in time of value 1. Let $\tilde{\mathcal{N}}_m^A$ denote the set of all the jobs on CPU m which are not a segment in time of value 1:

$$\tilde{\mathcal{N}}_m^A := \mathcal{N}_m^A \bigcap \left\{ k : \exists B_k^A < t' < E_k^A \;\; s.t. \;\; X_k^A(t') < 1 \right\}$$

Let us pick one member of that set (according to the assumption, the set is not empty), such that no other job on this set ends after her:

$$n \in \left\{ k : k \in \tilde{\mathcal{N}}_m^A \ \ and \ \ E_k^A \geq E_j^A \ \ \forall j \in \tilde{\mathcal{N}}_m^A \right\}.$$

Due to right-continuity of the work function $X_n^A(t)$,

$$\exists B_n^A < t_1 < E_n^A \ \ s.t. \ \ X_n^A(t_1) < 1 \Rightarrow$$
$$\exists \delta > 0, \epsilon > 0, t_2 > t_1 + \epsilon \ \ s.t.$$
$$\forall t_1 \leq t < t_2 \ \ X_n^A(t) < 1 - \delta.$$

Let $\tilde{\mathcal{N}}_{m,n}^A$ denote the set of jobs on CPU m, which are active while job n is active:

$$\tilde{\mathcal{N}}_{m,n}^A := \mathcal{N}_m^A \bigcap \{k : \exists B_n^A \leq t < E_n^A \ \ s.t. X_k^A(t) > 0\}.$$

According to lemma 3.23, the usage of CPU m is an interval in time of value 1 because allocation A optimizes function g. Hence, all the CPU time which is not devoted to job n must be devoted to other jobs, if job n has not ended yet (the CPU does not stand idle), and $\tilde{\mathcal{N}}_{m,n}^A \supsetneq \{n\}$.

Let $\tilde{I} = |\tilde{\mathcal{N}}_{m,n}^A|$. Let $i(k)$ be an ordinal number of job k in $\tilde{\mathcal{N}}_{m,n}^A$ according to its ending time, such that

$$E_{\{k:i(k)=1\}}^A \leq \ldots \leq E_{\{k:i(k)=\tilde{I}\}}^A.$$

$\forall k \in \tilde{\mathcal{N}}_{m,n}^A$ let us denote the CPU time job k got while job n was active by

$$s_{i(k)} := \int_{B_n^A}^{E_n^A} X_k^A(t) dt$$

Now let us define allocation A_2 such that:

$$\forall j \in \mathcal{M} \quad \mathcal{N}_j^{A_2} = \mathcal{N}_j^A$$

$$X_l^{A_2}(t) = \begin{cases} 1 & l \in \tilde{\mathcal{N}}_{m,n}^A \text{ and } t_{i(l)} \leq t < t_{i(l)+1} \\ 0 & l \in \tilde{\mathcal{N}}_{m,n}^A \text{ and } B_n^A \leq t < t_{i(l)} \\ 0 & l \in \tilde{\mathcal{N}}_{m,n}^A \text{ and } t_{i(l)} < t \\ X_l^A(t) & \text{otherwise} \end{cases}$$

where $t_1 = B_n^A$, $t_{\tilde{I}+1} = E_n^A$ and $\forall 1 < i(l) \leq \tilde{I}$, $t_{i(l)} = t_{i(l)-1} + s_{i(l)-1}$. Under allocation A_2, each job $k \in \tilde{\mathcal{N}}_{m,n}^A$ gets the same work as under allocation A until time B_n^A, and an additional work of $c_m s_{i(k)}$ from that time on, hence it gets the same work it got under allocation A. Hence, allocation A_2 fulfills the same vector of lengths $\vec{\theta}$ which A fulfills.

The order of ending times has not changed either. What has changed is the amount of work done for jobs with an ending time after job l, before job l ends. For job $l \in \tilde{\mathcal{N}}_{m,n}^A$, $l \neq n$, under both allocations all the jobs which end before job l must get their share of CPU time before job l ends. Under allocation A, some other jobs (at least job n) also get some work done until that time, and the following inequality is strict:

$$E_l^A > B_n^A + \sum_{p=1}^{i(l)-1} s_p.$$

Under A_2 on the other hand, while the ending time of job n remains the same $E_n^A = E_n^{A_2}$, all other ending times in $\tilde{\mathcal{N}}_{m,n}^A$ improve: $\forall l \in \tilde{\mathcal{N}}_{m,n}^A$, $l \neq n$, $E_l^{A_2} = B_n^A + \sum_{p=1}^{i(l)-1} s_p < E_l^A$.

Hence,

$$g(\vec{E}^{A_2}) > g(\vec{E}^A)$$

$$A \notin argmax_{a \in FUL(\vec{\theta})} g(\vec{E}^a)$$

in contradiction to the assumption. ■

Theorem 3.25 *If g is a regular social welfare function, then*

$$A \in argmax_{a \in FUL(\vec{\theta})} g(\vec{E}^a) \Rightarrow A \in EFF(\vec{\theta}).$$

Proof: Follows directly from lemma 3.23 and lemma 3.24. ■

Corollary 3.3.2 *If g is a regular social welfare function, and*

$$A \in argmax_{a \in FUL(\vec{\theta})} g(\vec{E}^a)$$

then A is Work-Function-Pareto efficient.

Proof: Assume in contradiction that A is not Work-Function-Pareto efficient. Then there is at least one job n, and an allocation $A' \in FUL(\vec{\theta})$, such that no other work function but n's is changed, and therefore:

$$\forall k \neq n \;\; E_k^A = E_k^{A'}$$

and still by changing n's work function alone,

$$E_n^A < E_n^{A'}.$$

Then the value of the social welfare function g is higher on allocation A':

$$g(\vec{E}^{A'}) > g(\vec{E}^A)$$

and A cannot optimize g:

$$A \notin argmax_{a \in FUL(\vec{\theta})} g(\vec{E}^a).$$

■

The institution's utility is a social welfare function. It is derived either from the vector of times agents receive their jobs in $\vec{T}$, or from another element of the status, such as $\vec{E}$. The first option is typical of an institution which benefits from completion of jobs, because the agents work for the institution to which the cluster belongs. The latter option is typical of institutions which sell computing power to external agents. Such an institution may wish, for example, to clear its cluster of jobs as soon as possible, rather than have the output delivered to the agents as soon as possible.

We will limit the discussion to social welfare functions of the first kind. We have already assumed the institution is indifferent to monetary transfers. Thus, the institution's utility is $g(\vec{T}^{Q(\infty)})$. An extension of this work may deal with more general institutional utilities, which are functions of $\vec{E}$ or of $\vec{X}$.

3.4 Motivating Example: A Straightforward Implementation

Had the institution known the lengths of the jobs $\vec{\theta}$, it could have chosen an allocation A to maximize the social welfare function. However, jobs' lengths are private information and whereas the

institution can ask for that information, agents may choose to provide false information to maximize their own utility.[16]

Following is an example where agents prefer to lie, for a given direct revelation mechanism. In this mechanism, the agents declare their lengths $\vec{b}$, and the institution allocates A so that it fulfills $\vec{b}$, and maximizes $g_{\sum}$.

Lemma 3.26 *An allocation which maximizes $g_{\sum}$ must be efficient.*

Proof: Follows directly from lemma 3.22 and theorem 3.25. ■

3.4.1 Example: The Need for Information

Let us examine the following situation:

- $N = 2$, $\vec{c} = (1)$.
- No payments are made.
- The static allocation is A, an allocation which fulfills $\vec{b}$ and maximizes $g(\vec{T}) = -(T_1 + T_2)$.
- $\forall n \in \mathcal{N}$, if $b_n < \theta_n$, $RN_n(E_n^A, 1, \theta_n)$ is applied. If $b_n > \theta_n$, no operator is applied. In other words, the final allocation is one which shares the same schedule as allocation A, but fulfills $\vec{\theta}$.

In the static allocation, $\vec{T} = \vec{E}^A$. In order to maximize g, given $\vec{b}$, assuming $\vec{b}$ truly represents $\vec{\theta}$, there are two possible allocations, both of which are efficient, Work-Function-Pareto-efficient, and fulfill $\vec{b}$. One of them, or both, maximize $g_{\sum}$ depending on whether $b_1 > b_2$ or the other way round. W.l.o.g., assume $b_1 \geq b_2$. Therefore, an optimal allocation is:

$$X_1^{A_{b_1>b_2}}(t) = \begin{cases} 0 & 0 \leq t < b_2 \\ 1 & b_2 \leq t < b_1 + b_2 \\ 0 & b_2 + b_1 \leq t \end{cases}$$

$$X_2^{A_{b_1>b_2}}(t) = \begin{cases} 1 & 0 < t \leq b_2 \\ 0 & b_2 < t < b_1 + b_2 \\ 0 & b_2 + b_1 < t \end{cases}$$

[16] Mechanisms that use the original type space as the signal space are referred to as 'Direct revelation' mechanisms. More generally, the institution may allow the use of an arbitrary signal space, and not necessarily jobs' lengths. However, due to the Revelation Principle [26], this restriction is without loss of generality.

and $\vec{T} = \vec{E}$.

Obviously, in our case where $M = 1$, every agent prefers to be scheduled first, and for that to happen, she needs to declare a type smaller than the other agent's. Hence, for both agents the best declaration would be a lie: $b_1 = b_2 = 0$, regardless of $\vec{\theta}$. Now assume $\theta_1 < \theta_2$ in our example. Note that agents lie, and the resulting allocation is not optimal.

3.5 The mechanism

A mechanism is a commitment of the institution on a particular course of action it would take in reaction to agents' declarations about their own lengths.[17] This course of action has three components:

1. Monetary transfers (prices).
2. A static allocation.
3. Job control triggers.

3.5.1 Prices

Definition 3.27 $\forall \vec{b} \in \Theta^N, g \in \mathcal{G}$, *A* **price vector** $\vec{P}(\vec{b}) \in \mathbb{R}^N$ *is a vector of functions* $P_n(\vec{b}) : \Theta^N \mapsto \mathbb{R}$, *each of which represents the amount of money agent n pays.*

If the price is negative, we may think of it as compensation the institution gives the agent.

3.5.2 A Static Allocation

Definition 3.28 $\forall \vec{b} \in \Theta^N, g \in \mathcal{G}$, *A* **static allocation** $o(\vec{b}, \vec{b}) : \Theta^N \mapsto \mathcal{A}$ *is an allocation which fulfills* $\vec{b}$ *and optimizes g.*

3.5.3 Job Control Triggers

When job n is completed (at time L_n), the institution knows that. If the job is not terminated before time E_n^A, then the institution knows that $\theta_n > b_n$. If $L_n < E_n^A$, the institution knows that $b_n > \theta_n$. It also knows the exact value of θ_n:

[17] See footnote 16 in this chapter.

$$\theta_n = \int_0^{L_n} X_n(t)dt$$

According to this information, job control tools are applied, subject to their existence on the operating system. An operator activated at time t_0 may not change work functions at times prior to t_0. At time t_0, the only information used in the activation of the operator, is the information available at that time. L_n is only known if job n has already terminated its execution.

In order to define the terms of activation of the job control tools, we require the following definitions:

Definition 3.29 *If $b_n \neq \theta_n$, agent n is* **lying**. *If $b_n > \theta_n$, agent n is* **lying upward**. *If $b_n < \theta_n$, agent n is* **lying downward**.

- **Systems With a Renice Tool**

  If $\theta_n > b_n$, the institution performs $RN_n(E_n^A, s_{renice}, \theta_n)$ according to some value of the percentage s_{renice} which is determined by the specific mechanism.

- **Systems With a Postpone Tool**

  If $\theta_n > b_n$, the institution performs $POST_n(E_n^A, s_{post} + E_n^{Q(0)} - E_n^{Q^-}, \theta_n)$, where $s_{post} > E_n^Q$ is a parameter depending on the specific mechanism. $E_n^{Q^-}$ stands for the time in which job n was supposed to be terminated according to the status when it began the execution.

  The term $E_n^{Q(0)} - E_n^{Q^-}$ comes to insure that a lying agent does not benefit from other agents lying upward.

  The RP operator which is one of the elements of $POST$ is also a simple degeneration: its effect is not the general effect of reducing the power to a certain percentage, but killing the job altogether. Therefore, the postpone tool might be available, while the Renice tool is not.

- **Systems With a Close-Gap Tool**

  When $\theta_n < b_n$, a termination happens at time t. The institution then reacts with

  $$CLOSE_m(L_n(t), E_n^A),$$

  where $n \in \mathcal{N}_m^A$.

- **Systems With an Early Release Tool**

  If $\theta_n = b_n$, the institution performs $EARLY_n$.

Note that the way we trigger them, the $POST$ and RN tools are replaceable, but cannot be combined, since they are activated by the same trigger.

Also note, that when RN or $CLOSE$ are activated, other jobs except the job which triggered the operator are affected:

Definition 3.30 *For any $n \in \mathcal{N}_m^A$, we denote by $F_n^A := \mathcal{N}_m^A \bigcap \{k : B_k^A \geq B_n^A\}$ the subset of jobs we refer to as job n's* **dependents***: the jobs, including n, which begin after job n on the CPU it runs on, under allocation A. Let $f_n = \left|F_n^A\right|$ denote job n's* **dependency number***: the number of jobs in set F_n^A.*

Definition 3.31 *A* **mechanism** *$Z = (o, \vec{P}, \vec{T})$ is composed of an output function, $o(\vec{x}, \vec{y}) : \Theta^{2N} \mapsto \mathcal{A}$, a vector of price functions $\vec{P} = \vec{P}(\vec{x})$ and a vector of output times $\vec{T}$ which is supported by $A = o(\vec{x}, \vec{y})$.*

A mechanism is a pre-defined protocol of actions, taken in a known environment, which is the rules of the game. The social welfare function g, and the price functions $\vec{P}(\vec{b})$ are common knowledge. Note that the price is a function of the declarations $\vec{b}$, and does not depend on $\vec{\theta}$.

3.5.4 The Game

We can now describe the N player game at the heart of our model. The game is held between the agents themselves, under the conditions determined by the institution.

1. Mechanism commitment: The institution commits to a given mechanism that is composed of:
   (a) $\forall \vec{b} \in \Theta^N, \;\; o(\vec{b}, \vec{b})$
   (b) $\forall \vec{b} \in \Theta^N, \;\; \vec{T}(\vec{b})$
   (c) $\forall \vec{b} \in \Theta^N, \;\; P(\vec{b})$
   (d) the available job control tools and their triggers: $\forall \vec{b}, \vec{\theta} \in \Theta^N, \;\; o(\vec{b}, \vec{\theta})$
2. Agents' types (lengths) are drawn and privately communicated to them.

3. Declaration: The agents declare their types $\vec{b} \in \Theta^N$.

4. Realization: Based on $\vec{b}$ and its earlier commitment, the institution decides on the static status $Q = (A, \vec{E}^A) \in FUL(\vec{b})$(where $A = o(\vec{b}, \vec{b})$), prices $\vec{P} = \vec{P}(\vec{b})$ and job control tool parameters: either s_{post} or s_{renice}, according to the job control tool in use. Note that at this point, the status is **delay free**: $\vec{T} = \vec{E}^A$.

5. Payment: The agents pay amounts $\vec{P}$.

6. Execution: The jobs are submitted to the cluster according to Q. Job control operators are applied according to the commitment in stage 1. We use the notation $\vec{T}(\vec{b}, \vec{\theta})$ to represent the output times vector resulting from a game in which the agents declare $\vec{b}$, and the real types are $\vec{\theta}$. We use the notation $\vec{U}(\vec{b}, \vec{\theta})$ to represent the vector of agents' utilities resulting from the same situation.

Note 3.5.1 *We currently assume that the agents who submit a bid cannot change their minds at this point and decide to withdraw. We discuss this possibility in chapter 7.5.*

3.6 Strategies

When facing the mechanism, an agent would decide upon certain courses of action, depending on her type. Some courses of action may always be at least as good as others, while some would be considered optimal in a narrower variety of cases.

Definition 3.32 $\forall \theta_n \in \Theta$*, agent real types, a* **strategy** *for agent n in a mechanism Z is a function* $S_n : \Theta \mapsto \Theta$*, such that* $S_n(\theta_n) = b_n$*, a declaration.*

By $S_{-n}(\theta_{-n}) = \times_{k \neq n} S_k(\theta_k)$ we denote the strategy tuple formed by all agents except n, when they use strategy S.

Definition 3.33 *Strategy S is a* **dominant strategy** *for agent n in a mechanism Z, if* $\forall \vec{\theta} \in \Theta^N, b_{-n} \in \Theta^{N-1}, b_n \in \Theta$,

$$U_n((S(\theta_n), b_{-n}), \vec{\theta}) \geq U_n((b_n, b_{-n}), \vec{\theta}).$$

Definition 3.34 *A strategy $S : \Theta \mapsto \Theta$ is in* **Ex-Post equilibrium** *if it is the best strategy against the other agents using the same strategy, regardless of what their types are: $\forall n \in \mathcal{N}, b_n \in \Theta, \vec{\theta} \in \Theta^N$,*

$$U_n((S_n(\theta_n), S_{-n}(\theta_{-n})), \vec{\theta}) \geq U_n((b_n, S_{-n}(\theta_{-n}), \vec{\theta})).$$

The term "Ex-Post Equilibrium" was introduced in Holzman et al. [18]. We use this term when agent n's strategy is optimal, assuming all others play their prescribed strategies. However, the Ex-Post equilibrium is stronger than the standard Bayesian equilibrium (which was introduced by Harsanyi [17]), as it is the best reply for agent n even after learning the realized types of her opponents. The Ex-Post equilibrium notion is weaker than the notion of dominance, because it hinges on agents compliance with the prescribed strategies, whereas a dominant strategy is optimal no matter how opponents act.

Chapter 4

Desired Mechanisms

In this chapter we present certain qualities of off-line scheduling mechanisms, by which we evaluate a mechanism. These qualities may be objective (either the mechanism has this quality or it does not), or they may be relative (senses in which a certain mechanism is preferable to others).

4.1 Incentive Compatibility (IC)

We wish to find a mechanism in which the agents are better off telling the truth. This way we verify that

$$o(\vec{b},\vec{b}) \in argmax_{FUL(\vec{b})} g(\vec{T}^{Q(0)}) \Rightarrow o(\vec{b},\vec{\theta}) \in argmax_{FUL(\vec{\theta})} g(\vec{T}^{Q(\infty)}).$$

The best sense in which the agents can be better off is the sense of dominant strategies.

Definition 4.1 *An arbitrary social welfare function $g \in \mathcal{G}$, is* **IC in dominant strategies** *in mechanism Z if $S(\theta_n) = \theta_n$ is a dominant strategy, when the static allocation optimizes the function g.*

A weaker notion of incentive compatibility is the following:

Definition 4.2 *An arbitrary social welfare function $g \in \mathcal{G}$, is* **IC in Ex-Post equilibrium** *in mechanism Z if $S(\theta_n) = \theta_n$ is in Ex-Post equilibrium, when the static allocation optimizes the function g.*

.

Definition 4.3 *Mechanism Z* **implements function g in Ex-Post Equilibrium** *if g is IC in Ex-Post equilibrium in mechanism Z.*

Mechanism Z **implements function g in dominant strategies** *if g is IC in dominant strategies in mechanism Z.*

4.2 Budget Considerations

As monetary transfers between the institution and the agents may be positive or negative, we look at the aggregate sum the institution gets (the **rent**), and check whether it is positive.

Definition 4.4 *A mechanism has* **positive rent** *if the institution never spends money: $\forall \vec{b} \in \Theta^N$, $\sum_{k \in \mathcal{N}} P_k(\vec{b}) \geq 0$.*

A more particular case of a positive rent (and a more desirable, since we assume the institution is indifferent about the payments the agents make), is a balanced budget.

Definition 4.5 *A mechanism is* **budget balanced** *if $\forall \vec{b} \in \Theta^N$, $\sum_{k \in \mathcal{N}} P_k(\vec{b}) = 0$.*

4.3 Safety Margins

Safety margins are the continuity of the utility of an agent with regard to the size of her lie, at the point of truth telling[1]. It is possible to demand that the agents tell the exact truth, if they actually know it. But as is often the case, agent n might have slightly perturbed information about its true length.

Definition 4.6 *A mechanism has* **upper safety margins** *if $\forall \epsilon > 0 \quad \exists \delta > 0$ s.t. $\forall n \in \mathcal{N}, \forall g \in \mathcal{G}, \forall \vec{b} \in \Theta^N, \vec{\theta} \in \Theta^N$*

$$\theta_n < b_n < \theta_n + \delta \Rightarrow \left| U_n(\vec{b}, (b_n, \theta_{-n})) - U_n(\vec{b}, \vec{\theta}) \right| < \epsilon.$$

[1] Note that this notion of continuity differs from the notion of continuity of the outcome function, which is introduced by Mount and Reiter [25], and is later used by Hurwicz [19]. The continuity of the outcome function relates to the continuity of the allocation with regard to the declarations. Obviously, we do not expect any off-line cluster scheduling mechanism to be continuous in that sense.

A mechanism has **lower safety margins** *if* $\forall \epsilon > 0 \exists \delta > 0$ *s.t.* $\forall n \in \mathcal{N}, \forall g \in \mathcal{G}, \forall \vec{b} \in \Theta^N, \vec{\theta} \in \Theta^N$

$$\theta_n - \delta < b_n < \theta_n \Rightarrow \left| U_n(\vec{b}, (b_n, \theta_{-n})) - U_n(\vec{b}, \vec{\theta}) \right| < \epsilon.$$

We say that a mechanism has **safety margins** *if it has both lower and upper safety margins.*

When a mechanism has one-sided safety margins (either upper safety margins or lower ones), and the agent only has distributional information d_n over θ_n, the strategy of telling the expectation value of the length of the job may be inferior to telling a diverted value, such as $\sup\{x : d_n(x) > 0\}$.

4.4 Fixed Prices

There is an advantage in setting the payment stage before the execution stage: since the monetary transfer may be from the agent to the institution (agents actually pay for the execution of the job), setting the payment stage before the execution enables the agent to change her mind and withdraw once she knows the price.

We design all the mechanisms we suggest such that their prices are fixed. This approach is not mandatory, as suggested by Nisan and Ronen [28]. In the spirit of their work, it is possible to build a mechanism for the off-line cluster scheduling problem, which maximizes the $g_{\sum}$ function. If the payment is postponed till after the jobs are executed, and their real lengths are verified, then the payment to agent n is composed of a compensation part, which equals the negation of her utility from execution, and a bonus part, which depends on the declarations of the other agents, and on her own real type. Nisan-Ronen's style payment function would be, in this case,

$$P_n(\vec{b}, \vec{\theta}) = -T_n(\vec{b}, \vec{\theta}) + \sum_{k \neq n} T_k(\vec{b}, (\theta_n, b_{-n}))$$

which would make the agent's utility

$$\begin{aligned} U_n(\vec{b}, \vec{\theta}) &= \\ -T_n(\vec{b}, \vec{\theta}) - P_n(\vec{b}, \vec{\theta}) &= -\sum_{k \neq n} T_k(\vec{b}, (\theta_n, b_{-n})) \end{aligned}$$

coincides with the $g_{\sum}$ social welfare function. In this mechanism the effect of θ_{-n} on the utility function of agent n is neutralized by verification, but the agents cannot know in advance the price they will have to pay.

4.5 Justness

An agent may feel more secure, if she knows that when she tells the truth, her output time will be no more than according to the static status. This is a quality of the mechanism:

Definition 4.7 *A mechanism is* **just** *if* $\forall n, \forall \vec{\theta}, \vec{b} \in \Theta^N$

$$b_n = \theta_n \;\Rightarrow\; T_n(\vec{b}, \vec{b}) \geq T_n(\vec{b}, \vec{\theta}).$$

In other words, if the mechanism is just, a truth teller is promised a minimal utility at payment time. The truth teller's utility will not be damaged when other agents' lies are revealed. This quality enables the institution to keep its commitments to the agents.

4.6 Scalability- Limitations on Input

A good mechanism accepts wide, possibly unlimited, ranges of real types, and manages to execute the jobs. It also works for large variances on the job lengths: different agents may have jobs which vary substantially.

4.7 Final Social Welfare

On Equilibrium, all agents report their real type, and the institution is able to optimize the final social welfare, which is equivalent to the initial social welfare. However, the social welfare of off-equilibrium situations carries importance as well. An off-equilibrium situation might be reached due to insufficient information, as described in section 4.3, or even due to irrational agents.

The social welfare function is not only a quality measurement: it is also the utility of the institution, (especially if the agents are workers of the institution): it benefits from the early completion of the jobs. A situation where a portion of the jobs is not completed is not acceptable.

Hence, we define a quality of final social welfare:

Definition 4.8 *Let* $Z^1 = (o^1, \vec{P}^1, \vec{T}^1)$, $Z^2 = (o^2, \vec{P}^2, \vec{T}^2)$, *be two mechanisms which optimize the same social welfare function* g. Z^1 *has a* **better final social welfare** *on* $\hat{\Theta}^N \subseteq \Theta^N$ *than* Z^2 *if* $\forall \vec{\theta} \in \hat{\Theta}, \vec{b} \in \Theta^N$,

$$\begin{aligned} o_1(\vec{b}, \vec{\theta}), o_2(\vec{b}, \vec{\theta}) &\in FUL(\vec{\theta}) \\ g(\vec{T}^{o_1(\vec{b},\vec{\theta})}) &\geq g(\vec{T}^{o_2(\vec{b},\vec{\theta})}). \end{aligned} \tag{4.1}$$

Chapter 5

A Special Case of a Social Welfare Function

In this chapter we focus on a particular, natural social welfare function: $g_{\sum} = \sum_{k \in \mathcal{N}} U_k$: the sum of utilities. We then turn to describe the Vickrey-Clarke-Groves mechanism, and view that in contrary to other environments, it is not effective in the off-line cluster scheduling environment. We end this chapter by proposing a mechanism which gives positive results for $g_{\sum}$ in this environment.

5.1 The Vickrey-Clarke-Groves Mechanism

5.1.1 Introduction

Consider an example where a single good is auctioned off. In that case, the Vickrey-Clarke-Groves (VCG) mechanism [37, 10, 16] implements $g = \sum_n W_n(o(\vec{b}), \theta_n)$ in dominant strategies, where $\vec{b}$ are the declarations, and $\vec{\theta}$ are the secrets: the real valuations of the good. $o(\vec{b})$ is the allocation (the decision: which of the agents gets the good), which depends only on the declarations, and W_n is the agent's utility from the allocation itself (not from the payment): if agent n gets the good, then her utility equals her valuation of the good. Otherwise, it equals zero. Thus, the agent's utility depends on the chosen allocation and on the agent's own type, only.

The major difference between an auction environment and the off-line cluster scheduling environment, is that the allocation in the former is based only on the declarations. The types in the auction environment need not be revealed, though some information about them leaks. The price the n-th agent pays is

$$P_n(\vec{b}) = -\sum_{k \neq n} V_k(\vec{b}, b_k).$$

Agent n's utility from the allocation combined with the money paid is

$$U_n(\vec{b}, \theta_n) = W_n(\vec{b}, \theta_n) + \sum_{k \neq n} W_k(\vec{b}, b_k)$$

so that when agent n is truth telling, her utility coincides with the social welfare function $g_{\sum}$,

$$U_n(\vec{b}, b_n) = \sum_{k=1}^{N} V_k(\vec{b}, b_k) \tag{5.1}$$

and is assured to be maximized by the mechanism, no matter what the other real types are. Thus, VCG ensures that $g_{\sum}$ is IC in dominant strategies.

5.1.2 A VCG Mechanism For Off-Line Cluster Scheduling

If we try to implement VCG in our environment, we face a problem. Here, the mechanism outputs a vector of work functions and a vector of output times. The agents' utility does not purely depend on the declarations $\vec{b}$, but also on the real lengths $\vec{\theta}$ of the other agents. If agent n lies and her job exceeds the limit of calculated allotted time $E_n^{Q^-}$, the beginning of the following (one or more) jobs may be delayed. If a job is shorter than declared, others may begin processing before scheduled.

The agent's utility from the allocation is $-T_n(\vec{b}, \vec{\theta})$. According to VCG, agent n should pay $P_n(\vec{b}) = \sum_{k \neq n} T_k(\vec{b}, \vec{b})$, and the overall utility in case agent n is truth telling, is

$$U_n(\vec{b}, (b_n, \theta_{-n})) = -T_n(\vec{b}, (b_n, \theta_{-n})) - \sum_{k \neq n} T_k(\vec{b}, \vec{b})$$

while the mechanism maximizes

$$-\sum_{k=1}^{N} T_k(\vec{b}, \vec{b}),$$

which does not coincide with a single agent's utility function. Therefore, the fact that the institution maximizes its own utility function does not guarantee the optimization of the agent's utility: It should be possible to find a combination of $\vec{b}, \vec{\theta}$ and $\vec{c}$, such that

$$U_n((b_n, b_{-n}), \vec{\theta}) > U_n((\theta_n, b_{-n}), \vec{\theta}).$$

In order to achieve an implementation in dominant strategies, it is essential to neutralize the effect of θ_{-n} on the utility function of agent n.

5.2 Hypotheses and Counter Examples

Could it be that the VCG payments are enough in order to insure the implementation of the $g_{\sum}$ social welfare function in dominant strategies? Could it be that no matter what set of well-defined job control tools we choose, a VCG mechanism would still implement the $g_{\sum}$ social welfare function?

We claim that it is not so: The strength in which the mechanism implements the social welfare function depends on the job control tools applied.

5.2.1 The Light VCG Mechanism: Counter Example 1

Let us introduce the **light VCG** mechanism:

- The prices paid reflect the sum of the other agents' utilities, according to $\vec{b}$:

$$P_n(\vec{b}) = -\sum_{k \neq n} U_k(\vec{b}, \vec{b}) = \sum_{k \neq n} T_k(\vec{b}, \vec{b})$$

- The only job control tool applied is $RN_n(E_n^A, 1, \theta_n)$, according to the triggers defined in subsection 3.5.3. Thus, jobs are run until their termination time, using the full CPU power. Of course, if a job exceeds the time allotted to it, it will cause a delay in the beginning time of its dependents (excluding itself, of course).

- The institution chooses the allocation such that $g_{\sum}(\vec{T})$ is maximal.

Under the light VCG mechanism, consider $\vec{\theta} = (1, 1.1)$, $\vec{c} = (1)$. Obviously, in the case of $M = 1$, the heuristic of performing the shortest job first (similar to the STF algorithm in Tanenbaum [34]),

maximizes $g_{\sum}$. Assume agent 2 declares $b_2 = 0.2 < \theta_2$, supposedly trying to beat agent 1 and get to be the first to run.

Let us compare agent 1's utility from declaring $b_1 = 0.1$, to her utility from telling the truth. In the first case, the static allocation would be:

$$X_1^{o((0.1,0.2),(0.1,0.2)))}(t)=\begin{cases} 1 & 0 \leq t < 0.1 \\ 0 & 0.1 \leq t \end{cases}$$

$$X_2^{o((0.1,0.2),(0.1,0.2)))}(t)=\begin{cases} 0 & 0 \leq t < 0.1 \\ 1 & 0.1 \leq t < 0.3 \\ 0 & 0.3 \leq t \end{cases}$$

and $\vec{T} = \vec{E}^{o((0.1,0.2),(0.1,0.2))}$.

Hence agent 1 would pay the price of $P_1 = E_2^{o((0.1,0.2),(0.1,0.2)))} = 0.3$. The final allocation would be

$$o((0.1, 0.2), \vec{\theta}) = RN_2(2.1, 0, 1.1) \circ RN_2(1.2, 1, 1.1)$$
$$\circ RN_1(1, 0, 1) \circ RN_1(0.1, 1, 1)(o((0.1, 0.2), (0.1, 0.2)))$$

which means the final work functions are

$$X_1^{o((0.1,0.2),\vec{\theta})}(t)=\begin{cases} 1 & 0 \leq t < 1 \\ 0 & 1 \leq t \end{cases}$$

$$X_2^{o((0.1,0.2),\vec{\theta})}(t)=\begin{cases} 0 & 0 \leq t < 1 \\ 1 & 1 \leq t < 2.1 \\ 0 & 2.1 \leq t \end{cases}$$

with $\vec{T} = \vec{E}^{o((0.1,0.2),\vec{\theta})} = (1, 2.1)$. The first agent's utility comes to

$$U_1((0.1, 0.2), \vec{\theta}) = -T_1((0.1, 0.2), \vec{\theta}) - T_2((0.1, 0.2), (0.1, 0.2)) = -1 - 0.3 = -1.3. \tag{5.2}$$

If agent 1 decides to tell the truth, the static allocation would be:

$$X_1^{o((1,0.2),(1,0.2))}(t)=\begin{cases} 0 & 0 \le t < 0.2 \\ 1 & 0.2 \le t < 1.2 \\ 0 & 1.2 \le t \end{cases}$$

$$X_2^{o((1,0.2),(1,0.2))}(t)=\begin{cases} 1 & 0 \le t < 0.2 \\ 0 & 0.2 \le t \end{cases}$$

With output times $\vec{T} = \vec{E}^{o((1,0.2),(1,0.2))} = (1.2, 0.2)$.

The price agent 1 pays would be smaller:

$$P_1 = E_2^{o((1,0.2),(1,0.2))} = 0.2.$$

The final allocation would be

$$o((1,0.2),\vec{\theta}) = RN_2(1.1,0,1.1) \circ RN_2(0.2,1,1.1)(o((1,0.2),(1,0.2)))$$

$$X_1^{o((1,0.2),\vec{\theta})}(t)=\begin{cases} 0 & 0 \le t < 1.1 \\ 1 & 1.1 \le t < 2.1 \\ 0 & 2.1 \le t \end{cases}$$

$$X_2^{o((1,0.2),\vec{\theta})}(t)=\begin{cases} 1 & 0 \le t < 1.1 \\ 0 & 1.1 \le t \end{cases}$$

with $\vec{T} = \vec{E}^{o((1,0.2),\vec{\theta})} = (2.1, 1.1)$. The first agent's utility comes to a value lower than in equation (5.2):

$$U_1((\theta_1,b_2),\vec{\theta}) = -T_1((\theta_1,b_2),\vec{\theta}) - T_2(\vec{b},\vec{b}) = -2.1 - 0.2 = -2.3.$$

So truth telling is not a dominant strategy, and therefore, VCG payments combined with an arbitrary set of job control tools, are not enough in order to implement $g_{\sum}$ in dominant strategies.

5.2.2 The Light VCG Mechanism: Counter Example 2

In subsection 5.2.1 we proved that truth telling is not a dominant strategy in the light VCG mechanism. The following example demonstrates that the light VCG mechanism does not implement $g_{\sum}$ in Ex-Post equilibrium, either.

Take the light VCG mechanism, on the case where $\vec{c} = \{1\}$, $\vec{\theta} = \{2, 1\}$. Assume agent 2 is truth telling, $b_2 = \theta_2$. If agent 1 tells the truth, $b_1 = \theta_1$, then both agents are truth telling, job 2 is executed first, and agent 1's utility comes to

$$U_1(\vec{\theta}, \vec{\theta}) = -T_1(\vec{b}, \vec{\theta}) - T_2(\vec{b}, \vec{b}) = -3 - 1 = -4.$$

If $b_1 = 0.5$, job number 1 is executed first, and

$$U_1((0.5, \theta_2), \vec{\theta}) = -T_1(\vec{b}, \vec{\theta}) - T_2(\vec{b}, \vec{b}) = -2 - 1.5 = -3.5.$$

Therefore, VCG payments are not enough in order to implement $g_{\sum}$ in Ex-Post equilibrium either.

5.3 VCG and Job control Tools

In this section we devise several methods which enable the implementation of $g_{\sum}$ in the off-line cluster scheduling environment.

5.3.1 The Harsh Punishment Mechanism

Let us introduce the **Harsh Punishment** mechanism:

- $\vec{P} = 0$: no prices are paid.
- Instead of the standard triggers we defined in subsection 3.5.3, the institution performs

  $$POST_n(min(E_n^A, L_n), \infty, \theta_n + \epsilon)$$

  for some positive value of ϵ, whenever $b_n \neq \theta_n$. This means that if an agent lies, she will never get her output, and the execution of her job will be stopped (postponed to infinity, or alternatively reniced to a zero share of the CPU), practically without altering the rest of the schedule.

- The chosen allocation optimizes $g_{\sum}$.

Obviously, this mechanism implements $g_{\sum}$ in dominant strategies, since truth telling is the only way to get the output of the job, thus achieving a finite utility $U_n = -T_n(\vec{b}, \vec{b}) > -\infty$.

The harsh mechanism is the trivial solution. While this mechanism is just, scalable, and its prices are known before the execution stage, it has no safety margins at all, and regarding its final social welfare it is the worst mechanism that could be.

5.3.2 Other Implementations of the $g_{\sum}$ Function

In the following lemmas, we prove that it is possible to implement $g_{\sum}$ using either of the two job control tools: *RN* or *POST*.

When discussing the case of $b_n < \theta_n$ in the *RN* based mechanism, job n's dependents are affected. Those agents' utilities depend upon agent n telling the truth, since the dependents' beginning time (excluding n's, of course) is increased if agent n is lying downward. Due to this dependency in the *RN* based mechanism, it is only possible to implement $g_{\sum}$ in Ex-Post Equilibrium. With *POST*, though, it is possible to implement $g_{\sum}$ in dominant strategies, since a downward lie does not affect truth telling dependents at all.

We will prove that the utility of agent n cannot increase beyond her utility while telling the truth, whether $b_n > \theta_n$ or $b_n < \theta_n$, in those two mechanisms: in the *POST* mechanism, while no assumptions on other agents' declarations are made, and in the *RN* mechanism, while other agents are assumed to be truth telling.

From the cases of $b_n < \theta_n$ we devise the valid ranges for the parameters s_{renice}, s_{post}: ranges for which the equilibria hold. It is left for the institution to choose the exact value for those parameters, within the permitted range. Though any value within the range will do the same when all agents are truth tellers, the choice of those values has an effect off-equilibrium on the final social welfare, as discussed in subsections 7.1 and 7.6.

Since later on we deal with social welfare functions other than $g_{\sum}$, and even use notations relevant to two social welfare functions in the same equations, let $\vec{T}_{\Sigma}, \vec{U}_{\Sigma}, o_{\sum}$ denote $\vec{T}, \vec{U}, o$ accordingly, in a system where the institution optimizes $g_{\sum}$.

Lemma 5.1 *In a system with RN, CLOSE and EARLY, where the social welfare function is* $g_{\sum}$, *and all other agents are truth telling, an agent cannot gain by declaring more than her real type.*

Proof: $\forall \vec{\theta}, \vec{b} \in \Theta^N$ s.t. $b_{-n} = \theta_{-n},\ b_n > \theta_n$, we wish to prove that

$$U_{\Sigma,n}(\vec{\theta}, \vec{\theta}) \geq U_{\Sigma,n}((b_n, \theta_{-n}), \vec{\theta}).$$

Agent n's utility is

$$U_{\Sigma,n}((b_n, \theta_{-n}), \vec{\theta}) = -T_{\Sigma,n}((b_n, \theta_{-n}), \vec{\theta}) - \sum_{k \neq n} T_{\Sigma,k}((b_n, \theta_{-n}), (b_n, \theta_{-n})).$$

All other agents are truth telling, so no alterations are made to A until time L_n, and the allocation is still efficient at that time, according to lemma 3.26. Hence,

$$L_n = E_n^A - \frac{b_n - \theta_n}{c_m},$$

where $n \in \mathcal{N}_m^A,\ A = o_{\sum}((b_n, \theta_{-n}), (b_n, \theta_{-n}).$

The final status is :

$$Q(\infty) = \prod_{k \neq n} EARLY_k \circ CLOSE_m(L_n, E_n^A)(Q).$$

So agent n will get her output in the same time as if her job's length was really b_n:

$$T_{\Sigma,n}((b_n, \theta_{-n}), \vec{\theta}) = T_{\Sigma,n}((b_n, \theta_{-n}), (b_n, \theta_{-n})). \tag{5.3}$$

Hence,

$$\begin{aligned} & U_{\Sigma,n}((b_n, \theta_{-n}), \vec{\theta}) &= \\ -T_{\Sigma,n}((b_n, \theta_{-n}), (b_n, \theta_{-n})) - \sum_{k \neq n} & T_{\Sigma,k}((b_n, \theta_{-n}), (b_n, \theta_{-n})) &= \\ & -\sum_{k=1}^{N} T_{\Sigma,k}((b_n, \theta_{-n}), (b_n, \theta_{-n})) &= g_{\sum}(\vec{E}^A). \end{aligned}$$

Since the institution optimizes $g_{\sum}(\vec{E}^A)$, the agent's utility function coincides with the social welfare function, in a similar way to the original VCG utility function in equation 5.1, which does not depend on other agents' real types. Hence, $b_n = \theta_n$ is the best declaration. ■

The following two lemmas will be used in the following proofs.

Lemma 5.2 $\forall \vec{x} \in \Theta^N, \forall y_n \in \Theta$,

$$\sum_{k\in\mathcal{N}} T_{\Sigma,k}((y_n, x_{-n}), (y_n, x_{-n})) - \sum_{k\in\mathcal{N}} T_{\Sigma,k}(\vec{x}, \vec{x}) \geq -f_n^A \frac{x_n - y_n}{c_m}$$

where $A = o_{\sum}((y_n, x_{-n}), (y_n, x_{-n}))$, $n \in \mathcal{N}_m^A$ *and* f_n^A *is defined according to definition 3.30.*

Proof: Let $A = o_{\sum}((y_n, x_{-n}), (y_n, x_{-n}))$. Let us compare two allocations which share the schedule of allocation A: A and A'. Let A' fulfill $\vec{x}$. A' can be created from A by applying

$$\prod_{k\in F_n^A} RN(L_k, 1, x_k).$$

Then we can bound :

$$\sum_{k\in\mathcal{N}} T_{\Sigma,k}(\vec{x}, \vec{x}) \leq \sum_{k\in\mathcal{N}} T_{\Sigma,k}((y_n, x_{-n}), (y_n, x_{-n})) + f_n^A \frac{x_n - y_n}{c_m}$$

where $n \in \mathcal{N}_m^A$. ■

When the total work to be done is bigger, it is impossible for a system which optimizes $g_{\sum}$ to reach a larger value for $g_{\sum}(\vec{E})$, as we prove in the following lemma.

Lemma 5.3 $\forall \vec{\theta} \in \Theta^N$, *if* $\theta_n < b_n \in \Theta$, *then*

$$\sum_{k\in\mathcal{N}} T_{\Sigma,k}(\vec{\theta}, \vec{\theta}) \leq \sum_{k\in\mathcal{N}} T_{\Sigma,k}((b_n, \theta_{-n}), (b_n, \theta_{-n})).$$

Proof: Using lemma 5.2, we can bound :

$$\begin{aligned} \sum_{k\in\mathcal{N}} T_{\Sigma,k}(\vec{\theta}, \vec{\theta}) &\leq \\ \sum_{k\in\mathcal{N}} T_{\Sigma,k}((b_n, \theta_{-n}), (b_n, \theta_{-n})) - f_n^A \frac{b_n - \theta_n}{c_m} &\leq \\ \sum_{k\in\mathcal{N}} T_{\Sigma,k}((b_n, \theta_{-n}), (b_n, \theta_{-n})) & \end{aligned}$$

where $A = o_{\sum}((b_n, \theta_{-n}), (b_n, \theta_{-n}))$, $n \in \mathcal{N}_m^A$. ■

Lemma 5.4 *In a system with RN, CLOSE and EARLY, where the social welfare function is* $g_{\sum}$, *and all other agents are truth telling, an agent cannot gain by declaring less than her real type.*

Proof: When agent n lies such that $b_n < \theta_n$, the institution performs $RN_n(E_n^A, s_{renice}, \theta_n)$, where the value of s_{renice} is a free parameter, left for the institution to define. $\forall \vec{\theta}, \vec{b} \in \Theta^N$ s.t. $b_{-n} =$

$\theta_{-n},\ b_n < \theta_n$, let us calculate a range of values for s_{renice}, sufficient to insure that lying upward is not beneficial. s_{renice} should be s.t. $\forall 0 < b_n < \theta_n$, it holds that:

$$\begin{aligned} U_{\Sigma,n}(\vec{\theta},\vec{\theta}) &\geq U_{\Sigma,n}((b_n,\theta_{-n}),\vec{\theta}) \\ -T_{\Sigma,n}(\vec{\theta},\vec{\theta}) - \sum_{k\neq n} T_{\Sigma,k}(\vec{\theta},\vec{\theta}) &\geq -T_{\Sigma,n}((b_n,\theta_{-n}),\vec{\theta}) - \sum_{k\neq n} T_{\Sigma,k}((b_n,\theta_{-n}),(b_n,\theta_{-n})). \end{aligned} \quad (5.4)$$

Let $A := o_{\sum}((b_n,\theta_{-n}),(b_n,\theta_{-n})) = o_{\sum}(\vec{b},\vec{b})$. Due to the use of $RN_n(E_n^A, s_{renice}, \theta_n)$,

$$T_{\Sigma,n}((b_n,\theta_{-n}),\vec{\theta}) = T_{\Sigma,n}((b_n,\theta_{-n}),(b_n,\theta_{-n})) + \frac{\theta_n - b_n}{s_{renice} c_m} \quad (5.5)$$

where m is s.t. $n \in \mathcal{N}_m^A$. Hence, after substituting equation (5.5) in equation (5.4), we can tell that we need to find a range for s_{renice}, s.t. it satisfies

$$\sum_{k\in\mathcal{N}} T_{\Sigma,k}(\vec{\theta},\vec{\theta}) \leq \frac{\theta_n - b_n}{s_{renice} c_m} + \sum_{k\in\mathcal{N}} T_{\Sigma,k}((b_n,\theta_{-n}),(b_n,\theta_{-n})) \quad (5.6)$$

Using lemma 5.2 together with equation (5.6), we get that it is sufficient for s_{renice} to satisfy

$$s_{renice} \leq \frac{1}{f_n^A}.$$

Note that the value of s_{renice} depends only on the values of $\vec{b}$, so it can be declared before the execution stage, either for each job separately (by setting s_{renice} for job n as $\frac{1}{f_n^A}$), or together for all jobs, according to $s_{renice} = min_{k\in\mathcal{N}} \frac{1}{f_n^A}$. ■

Theorem 5.5 *In a system with RN, CLOSE and EARLY, it is possible to implement $g_{\sum}$ in Ex-Post equilibrium.*

Proof: Follows immediately from lemma 5.1 and lemma 5.4. ■

An Implementation of $g_{\sum}$ in Dominant strategies

Lemma 5.6 *In a system with POST, CLOSE and EARLY, where the social welfare function is $g_{\sum}$, an agent cannot gain by declaring more than her real type.*

Proof: $\forall \vec{\theta},\vec{b} \in \Theta^N$, s.t. $b_n > \theta_n$, we wish to prove that

$$U_n((\theta_n,b_{-n}),\vec{\theta}) - U_n(\vec{b},\vec{\theta}) \geq 0.$$

Let us concentrate on Q_m, the status of CPU m. Let $n \in \mathcal{N}_m^A$, $A = o_{\sum}(\vec{b}, \vec{b})$. Let $\mathcal{N}_{-n}^A := \mathcal{N}_m^A \backslash \{n\}$. Since job control operators, which are invoked due to a lie of a certain agent, affect only the status of the CPU that agent was allocated to, the dynamic status of CPU m[1] depends only on the static status for CPU m, and a combination of operators on that CPU or on jobs on that CPU.

$$Q_{m,dynamic} = \prod_{k \in \mathcal{N}_{-n}^A} \{O_k, CLOSE_m(L_k, E_k^{Q^{k,-}})\}(Q)$$

where O_k stands for either $POST_k$ or $EARLY_k$, and $Q^{k,-}$ stands for the dynamic status at the time in which job k begins its execution.

$$T_{\Sigma,n}^{Q_{final}} = T_{\Sigma,n}(\vec{b}, \vec{\theta}) = T_{\Sigma,n}(\vec{b}, \vec{b}). \tag{5.7}$$

Equation (5.7) holds since: $\forall k$, $POST_k$ affects only E_k and T_k. $EARLY_k$ affects only T_k. None of these operators change B_n, E_n or T_n. The $CLOSE$ operators performed before B_n^A will change B_n and E_n, but not T_n. Even if E_n has changed by time L_n, T_n will only change if $EARLY_n$ is applied. $EARLY_n$ is not applied, of course, since $b_n \neq \theta_n$. $POST_n$, which can also affect T_n is not applied either, since $b_n < \theta_n$.

Had agent n told the truth, $EARLY_n$ would have been applied. In that case we would get:

$$T_{\Sigma,n}((\theta_n, b_{-n}), \vec{\theta}) \leq T_{\Sigma,n}((\theta_n, b_{-n}), (\theta_n, b_{-n})) \tag{5.8}$$

and using equation (5.7) and equation (5.8) we get

$$\begin{aligned} U_{\Sigma,n}((\theta_n, b_{-n}), \vec{\theta}) - U_{\Sigma,n}(\vec{b}, \vec{\theta}) &= \\ -T_{\Sigma,n}((\theta_n, b_{-n}), \vec{\theta}) + T_{\Sigma,n}(\vec{b}, \vec{\theta}) - \sum_{k \neq n} T_{\Sigma,k}((\theta_n, b_{-n}), (\theta_n, b_{-n})) + \sum_{k \neq n} T_{\Sigma,k}(\vec{b}, \vec{b}) &\geq \\ - \sum_{k \in \mathcal{N}} T_{\Sigma,k}((\theta_n, b_{-n}), (\theta_n, b_{-n})) + \sum_{k \in \mathcal{N}} T_{\Sigma,k}(\vec{b}, \vec{b}) &> 0. \end{aligned}$$

The last inequality holds due to lemma 5.3. ■

Note that if agent n lies downward, the $POST$ job control tool does not permit a change in B_k, not in E_k for job k, which is allocated to the same CPU as n, at a later time. On the other hand,

[1]The status of CPU m was defined in definition 3.10.

when job n depends on jobs whose agents lied upward, the performed $CLOSE$ operator decreases B_n and E_n. The decreasing of T_n to match E_n depends on the institution performing $EARLY_n$. Thus, for every set of real types and declarations, $EARLY_n$ may decrease T_n in a different amount of time: the possible gained time, defined as follows:

Definition 5.7 $\forall \vec{b} \in \Theta^N,\ \theta_{-n} \in \Theta^{N-1},\ g \in \mathcal{G},\ A = o(\vec{b}, \vec{b}), n \in \mathcal{N},$ *let*

$$D_n^A(\vec{b}, \theta_{-n}) := T_n(\vec{b}, \vec{b}) - T_n(\vec{b}, (b_n, \theta_{-n}))$$

$D_n^A(\vec{b}, \theta_{-n})$ *is the* **possible gained time** *when the institution optimizes $g(\vec{T})$, the agents declare $\vec{b}$ and the other agents' real types are $\vec{\theta}$. $D_n^A(\vec{b}, \theta_{-n})$ is the amount of time by which the beginning time of job n can be made earlier due to lies of fellow agents.*

Note that $D_n^A(\vec{b}, \theta_{-n}) \geq 0$. Of course, the value of $D_n^A(\vec{b}, \theta_{-n})$ actually depends only on the real length of the subset of jobs that job n depends on.

Lemma 5.8 *In a system with $POST$, $CLOSE$ and $EARLY$, where the social welfare function is $g_{\sum}$, an agent cannot gain by declaring less than her real type.*

Proof: $\forall \vec{\theta}, \vec{b} \in \Theta^N$, s.t. $b_n < \theta_n$, we wish to find a range for s_{post}, which is to be determined by the mechanism, s.t.

$$U_n(\vec{\theta}, \vec{\theta}) \geq U_n((b_n, \theta_{-n}), \vec{\theta}).$$

Let $n \in \mathcal{N}_m^A,\ A = o(\vec{b}, \vec{b})$. Using the same notation as in the proof of lemma 5.6,

$$Q_{m,dynamic} = \prod_{k \in \mathcal{N}_{-n}^A \text{ and } B_k^A > E_n^A} \{O_k, CLOSE_m(L_k, E_k^{Q^{k,-}})\}$$
$$\circ POST_n(E_n^A, s, \theta_n) \circ \prod_{k \in \mathcal{N}_{-n}^A \text{ and } E_k^A < B_n^A} \{O_k, CLOSE_m(L_k, E_k^{Q^{k,-}})\}(Q)$$

As in the proof of lemma 5.6, $POST_k$ and $EARLY_k$ affect only the times of the agent k. $CLOSE$ operators performed on the set $\mathcal{N}_{-n}^A \bigcap \{k : E_k^A < B_n^A\}$ will decrease B_n and E_n in the same amount, independently of the value of θ_n. The only differences in agent n's utility, between the case where agent n is truth telling and the case of $b_n < \theta_n$, are due to the possible gained time and the affect of $POST_n$.

We seek a range for s_{post} s.t.

$$\begin{aligned} U_{\Sigma,n}((\theta_n, b_{-n}), \vec{\theta}) - U_{\Sigma,n}(\vec{b}, \vec{\theta}) &= \\ T_{\Sigma,n}(\vec{b}, \vec{\theta}) - T_{\Sigma,n}((\theta_n, b_{-n}), \vec{\theta}) + \sum_{k \neq n} T_{\Sigma,k}(\vec{b}, \vec{b}) - \sum_{k \neq n} T_{\Sigma,k}((\theta_n, b_{-n}), (\theta_n, b_{-n})) &\geq 0. \end{aligned}$$

According to the definition of $POST_n$:

$$T_{\Sigma,n}(\vec{b}, \vec{\theta}) = T_{\Sigma,n}(\vec{b}, \vec{b}) + \frac{\theta_n - b_n}{c_m} + s_{post} + D_n^A(\vec{b}, \theta_{-n}). \tag{5.9}$$

$$\begin{aligned} -T_{\Sigma,n}((\theta_n, b_{-n}), \vec{\theta}) = \\ -T_{\Sigma,n}((\theta_n, b_{-n}), (\theta_n, b_{-n})) - D_n^{A'}((\theta_n, b_{-n}), \theta_{-n}) \geq \\ -T_{\Sigma,n}((\theta_n, b_{-n}), (\theta_n, b_{-n})) \end{aligned} \tag{5.10}$$

where $A' = o_{\sum}((\theta_n, b_{-n}), (\theta_n, b_{-n}))$.

By definition,

$$\begin{aligned} U_{\Sigma,n}((\theta_n, b_{-n}), \vec{\theta}) - U_{\Sigma,n}(\vec{b}, \vec{\theta}) = \\ -T_{\Sigma,n}((\theta_n, b_{-n}), \vec{\theta}) + T_{\Sigma,n}(\vec{b}, \vec{\theta}) \\ - \sum_{k \neq n} T_{\Sigma,k}((\theta_n, b_{-n}), (\theta_n, b_{-n})) + \sum_{k \neq n} T_{\Sigma,k}(\vec{b}, \vec{b}). \end{aligned} \tag{5.11}$$

Using equation (5.10) we get

$$\begin{aligned} -T_{\Sigma,n}((\theta_n, b_{-n}), \vec{\theta}) - \sum_{k \neq n} T_{\Sigma,k}((\theta_n, b_{-n}), (\theta_n, b_{-n})) \geq \\ - \sum_{k \in \mathcal{N}} T_{\Sigma,k}((\theta_n, b_{-n}), (\theta_n, b_{-n})) \end{aligned}$$

and using equation (5.9):

$$\begin{aligned} T_{\Sigma,n}(\vec{b}, \vec{\theta}) + \sum_{k \neq n} T_{\Sigma,k}(\vec{b}, \vec{b}) \geq \\ \sum_{k \in \mathcal{N}} T_{\Sigma,k}(\vec{b}, \vec{b}) + \frac{\theta_n - b_n}{c_m} + s_{post}. \end{aligned}$$

When substituting back into equation (5.11) we get

$$\begin{aligned} U_{\Sigma,n}((\theta_n, b_{-n}), \vec{\theta}) - U_{\Sigma,n}(\vec{b}, \vec{\theta}) \geq \\ \frac{\theta_n - b_n}{c_m} + s_{post} - \sum_{k \in \mathcal{N}} T_{\Sigma,k}((\theta_n, b_{-n}), (\theta_n, b_{-n})) + \sum_{k \in \mathcal{N}} T_{\Sigma,k}(\vec{b}, \vec{b}) \end{aligned}$$

Substituting lemma 5.2, we get :

$$U_{\Sigma,n}((\theta_n, b_{-n}), \vec{\theta}) - U_{\Sigma,n}(\vec{b}, \vec{\theta}) \geq (\theta_n - b_n) \left(\frac{1}{c_m} - \frac{f_n^{A'}}{c_{m'}} \right) + s_{post}.$$

where $A' = o_{\sum}((b_n, \theta_{-n}), (b_n, \theta_{-n}))$, $n \in \mathcal{N}_{m'}^{A'}$.

So it is sufficient for s_{post} to fulfill the following condition:

$$s_{post} \geq (\theta_n - b_n) \left(\frac{N}{\min_{j \in \mathcal{M}} c_j} - \frac{1}{c_m} \right) \geq (\theta_n - b_n) \left(\frac{f_n^{A'}}{c_{m'}} - \frac{1}{c_m} \right). \tag{5.12}$$

Since the expression $(\theta_n - b_n) \left(\frac{N}{\min_{j \in \mathcal{M}} c_j} - \frac{1}{c_m} \right)$is non-negative, we present one of the following limitations to insure that s_{post} fulfills the condition:

1. Let $H_{diff} \in \mathbb{R}_+$ be an arbitrary parameter of the cluster. If $\theta_n - b_n > H_{diff}$, the lie is too large, and the execution is stopped (a limitation on the downward lie).

   Before the execution stage, the institution needs to set s_{post} s.t.

   $$s_{post} \geq H_{diff} \left(\frac{N}{\min_{j \in \mathcal{M}} c_j} - \frac{1}{c_m} \right). \tag{5.13}$$

2. Let $H_{diff} \in \mathbb{R}_+$ be an arbitrary parameter of the cluster. θ_n is limited: execution is stopped when amount of work H_{abs} is performed on a single job. The institution sets before the execution stage s_{post} s.t.

   $$s_{post} \geq (H_{abs} - b_n) \left(\frac{N}{\min_{j \in \mathcal{M}} c_j} - \frac{1}{c_m} \right). \tag{5.14}$$

■

Theorem 5.9 *In a system with POST, CLOSE and EARLY, it is possible to implement $g_{\sum}$ in dominant strategies.*

Proof: Follows immediately from lemma 5.6 and lemma 5.8. ■

5.4 Are Monetary Transfers Really Needed?

In this section we verify the necessity of the monetary transfers in the class of mechanisms which we devised.

We have proved that a combination of job control tools and monetary transfers is sufficient to implement $g_{\sum}$. Would a mechanism which uses job control tools the same way, but does not allow monetary transfers, implement $g_{\sum}$ as well?

Example 5.4.1 *In a mechanism where*

- *the available job control tools are POST, EARLY and CLOSE, triggered according to subsection 3.5.3.*
- $\vec{P} = 0$.
- *the institution maximizes* $g_{\sum}$

truth telling is not IC in Ex-Post equilibrium.

Proof: Take the case of $c = \{1, 1.5\}$, $\vec{\theta} = \{1, 1.01\}$, and assume $b_2 = \theta_2$. If agent 1 tells the truth, job 1 is located on c_1, and

$$U_{\Sigma,1}(\vec{b}, \vec{\theta}) = -T_{\Sigma,1}(\vec{b}, \vec{\theta}) = -\frac{1}{1} + 0 = -1.$$

On the other hand, if agent 1 declares $b_1 = 1.1$, job 1 is located on c_2, and

$$U_{\Sigma,1}(\vec{b}, \vec{\theta}) = -T_{\Sigma,1}(\vec{b}, \vec{\theta}) = -\frac{1.1}{1.5} + 0 > -1.$$

That happens due to $EARLY_1$ not being activated: since agent 1 lied, she does not get her output as soon as she can, but rather as if θ_1 indeed equaled b_1. ■

Chapter 6

A General Social Welfare Function

6.1 introduction

In this chapter we propose a mechanism which implements general social welfare functions in the off-line cluster scheduling environment.

$g = -\sum T_n$ is usually a practical social welfare function to implement, since the throughput, defined as $\frac{N}{\sum_{n=1}^{N} T_n}$, increases when $g = -\sum_{n=1}^{N} T_n$ increases. The throughput is a good measurement for the average efficiency[1] of the use of the resource.

And yet, when the important thing is not average throughput, but dealing with bursts of tasks, the institution often cares to implement a social welfare function other than $g_{\sum}$. A common example is the make-span function, as indicated for example by Abdekhodaee and Wirth[1] by Dodin and Elimam [12].

Definition 6.1 $\forall l \in 1..N$, *the* **make-span of order** l *is a function* $makespan^l : \mathbb{R}_+^N \mapsto \mathbb{R}_+$ *which is defined as*

$$makespan^l(\vec{T}) = - \max_{k_1 \neq k_2 \ldots \neq k_l \in \mathcal{N}} \sum_{i=1}^{l} T_{k_i}.$$

The common social welfare function is $makespan^1$, which is actually $-max_{k \in \mathcal{N}} T_k$. Optimizing this function means choosing the allocation such that the lowest utility among all agents is optimized.

[1] The term "efficiency" comes here in its everyday meaning, without connection to definition 3.6 of efficient allocations.

Since several allocations may optimize the make-span function, a common tie-breaker (instead of a lottery, for example) is the lexical max function.

Definition 6.2 *$\forall \vec{b} \in \Theta^N$, let*

$$ms^0(\vec{b}) = FUL(\vec{b}),$$

and $\forall l \in 1..N$ let

$$ms^l(\vec{b}) = \{A : \vec{E}^A \in argmax_{A' \in ms^{l-1}(\vec{b})} makespan^l(\vec{E}^{A'})\}$$

$\forall A \in FUL(\vec{b})$ let us define the **lexical-max** *function, $lexicalmax : FUL(\vec{b}) :\mapsto \mathbb{R}_+$ as*

$$lexicalmax(A) = \sup\{l : A \in ms^l(\vec{b})\}$$

In other words, the set $ms^0(\vec{b})$ consists of all the allocations which fulfill $\vec{b}$. $ms^1(\vec{b}) \subseteq ms^0(\vec{b})$ is a subset of allocations which minimize the time of the last job to be completed. $ms^2(\vec{b}) \subseteq ms^1(\vec{b})$ consists of the allocations which, in addition to minimizing the time of the last job to finish, minimize the time of the job to finish before last, and so forth.

Lexical max is a good social welfare function mainly when the institution is the employer of the agents and not just the cluster owner.[2] Another example of a social welfare function might be a weighted make-span, or a weighted sum of utilities.

6.2 The Extended VCG (EVCG) Mechanism

We already know (from theorems 5.5 and 5.9)that the VCG payment function, combined with certain job control tools, enables the implementation of the $g_{\sum}$ function. Let us try and optimize a general social welfare function when picking the allocation, while compensating the agent. We could pretend that we offer the agent to participate in a mechanism which optimizes $g_{\sum}$, and then we offer her a monetary compensation for changing the allocation to one that optimizes another social welfare function. Under that spirit, let us examine the **extended VCG payment function**:

$$P_n(\vec{b}) = -T_n(\vec{b}, \vec{b}) + T_{\Sigma,n}(\vec{b}, \vec{b}) + \sum_{k \neq n} T_{\Sigma,k}(\vec{b}, \vec{b})$$

[2]Lexical max is less demanding than a precedence constraint, which Azar and Epstein [5] deal with. Under a precedence constraint, certain (specific) jobs must end before another can begin, and the institutional utility depends on the time the last one ended in. Under lexical max the order within the group of jobs does not matter, but the institutional utility does depend on the completion of the whole set of jobs.

$$= -T_n(\vec{b}, \vec{b}) + \sum_{k=1}^{N} T_{\Sigma,k}(\vec{b}, \vec{b}). \tag{6.1}$$

The extended VCG payment function is composed of three terms:

1. $-T_n(\vec{b}, \vec{b})$ a compensation for the time spent, assuming all agents are truth telling.
2. $T_{\Sigma,n}(\vec{b}, \vec{b})$ the negation of the time that would have been spent, assuming all agents are truth telling, had the social welfare function been g_Σ.
3. $\sum_{k \neq n} T_{\Sigma,k}(\vec{b}, \vec{b})$ the payment that would have been inflicted, had the social welfare function been $g_{\sum}$.

Note 6.2.1 *Assuming all agents are truth telling, an agent's utility from any social welfare function, with the extended VCG payment, is identical to her utility from a system with a social welfare function of* $g_{\sum}$,

$$U_n(\vec{\theta}, \vec{\theta}) = U_{\Sigma,n}(\vec{\theta}, \vec{\theta}).$$

Definition 6.3 *We refer to a mechanism which uses the extended VCG payment function as a* **extended VCG mechanism**, *and in short:* **EVCG**.

Lemma 6.4 $\forall n \in \mathcal{N}$,*under the EVCG mechanism, in a system with* $g \in \mathcal{G}$, $CLOSE$ *and* $EARLY$, *where one of the following assumptions holds:*

1. *The system makes use of* RN *and everybody else is truth telling* $b_{-n} = \theta_{-n}$.
2. *The system makes use of* $POST$.

agent n *cannot gain by declaring more than her real type.*

Proof: $\forall \vec{\theta}, \vec{b} \in \Theta^N$, s.t. $b_n > \theta_n$, we wish to prove that given either one of the assumptions,

$$U_n((\theta_n, b_{-n}), \vec{\theta}) \geq U_n(\vec{b}, \vec{\theta}). \tag{6.2}$$

Let us develop the right-hand side of equation (6.2). From the definition of the EVCG payment function

$$U_n(\vec{b}, \vec{\theta}) = -T_n(\vec{b}, \vec{\theta}) + T_n(\vec{b}, \vec{b}) - \sum_{k=1}^{N} T_{\Sigma,k}(\vec{b}, \vec{b}). \tag{6.3}$$

Under assumption 1, in the same way that equation (5.3) is reached in the proof of lemma 5.1,

$$T_n(\vec{b},\vec{\theta}) = T_n(\vec{b},\vec{b}). \tag{6.4}$$

Under assumption 2, equation (6.4) holds due to the same arguments used in the proof of lemma 5.6, for equation (5.7).

Substituting equation (6.4) into equation (6.3) yields, under both assumptions,

$$U_n(\vec{b},\vec{\theta}) = -\sum_{k=1}^{N} T_{\Sigma,k}(\vec{b},\vec{b}). \tag{6.5}$$

Now let us develop the left-hand side of equation (6.2). When assumption 1 is valid, lemma 6.2.1 means that

$$U_n((\theta_n, b_{-n}),\vec{\theta}) = -\sum_{k=1}^{N} T_{\Sigma,k}((\theta_n, b_{-n}),(\theta_n, b_{-n})). \tag{6.6}$$

When assumption 2 is valid, if agent n tells the truth, her output time may only decrease relative to the static status: other agents, on whose jobs n depends, will decrease her output time if an upward lie by them is revealed. A downward lie by the same agents will not change agent n's output time.

$$T_n((\theta_n, b_{-n}),\vec{\theta}) \leq T_n((\theta_n, b_{-n}),(\theta_n, b_{-n})) \tag{6.7}$$

Since according to the definition of the EVCG payment function,

$$U_n((\theta_n, b_{-n}),\vec{\theta}) =$$
$$\Big\{-T_n((\theta_n, b_{-n}),\vec{\theta}) + T_n((\theta_n, b_{-n}),(\theta_n, b_{-n}))\Big\} - \sum_{k=1}^{N} T_{\Sigma,k}((\theta_n, b_{-n}),(\theta_n, b_{-n})),$$

equation (6.7) means that

$$U_n((\theta_n, b_{-n}),\vec{\theta}) \geq -\sum_{k=1}^{N} T_{\Sigma,k}((\theta_n, b_{-n}),(\theta_n, b_{-n})). \tag{6.8}$$

Using equation (6.6) when assumption 1 is true, and using equation (6.8) when assumption 2 is true, and using equation (6.5) on both cases, we get

$$\begin{aligned} U_n((\theta_n, b_{-n}),\vec{\theta}) - U_n(\vec{b},\vec{\theta}) &\geq \\ -\sum_{k=1}^{N} T_{\Sigma,k}((\theta_n, b_{-n}),(\theta_n, b_{-n})) + \sum_{k=1}^{N} T_{\Sigma,k}(\vec{b},\vec{b}) &\geq 0. \end{aligned}$$

where lemma 5.3 is used for the last inequality. ■

6.3 An Implementation of a General Social Welfare Function in Ex-Post Equilibrium

Lemma 6.5 *Under the EVCG mechanism, in a system with $g \in \mathcal{G}$, RN, CLOSE and EARLY, where all other agents are truth telling, an agent cannot gain by declaring less than her real type.*

Proof: Since $b_n < \theta_n$, $RN_n(E_n^A, s_{renice}, \theta_n)$ is performed, where the value of s_{renice} can be defined by the mechanism, and $A = o((b_n, \theta_{-n}), (b_n, \theta_{-n}))$.

$\forall \vec{\theta}, \vec{b} \in \Theta^N$, s.t. $b_n < \theta_n$, $b_{-n} = \theta_{-n}$, we wish to find a range for s_{renice} s.t.

$$U_n(\vec{\theta}, \vec{\theta}) - U_n((b_n, \theta_{-n}), \vec{\theta}) \geq 0.$$

Due to the operator $RN(E_n^A, s_{renice}, \theta_n)$,

$$T_n((b_n, \theta_{-n}), \vec{\theta}) - T_n((b_n, \theta_{-n}), (b_n, \theta_{-n})) = \frac{\theta_n - b_n}{s_{renice} c_m}$$

where $n \in \mathcal{N}_m^A$.

Using also note 6.2.1 we get

$$\begin{aligned} U_n(\vec{\theta}, \vec{\theta}) - U_n((b_n, \theta_{-n}), \vec{\theta}) &= \\ -\sum_{k=1}^{N} T_{\Sigma,k}(\vec{\theta}, \vec{\theta}) & \\ + \left\{ T_n((b_n, \theta_{-n}), \vec{\theta}) - T_n((b_n, \theta_{-n}), (b_n, \theta_{-n})) \right\} + \sum_{k=1}^{N} T_{\Sigma,k}((b_n, \theta_{-n}), (b_n, \theta_{-n})) &= \\ -\sum_{k=1}^{N} T_{\Sigma,k}(\vec{\theta}, \vec{\theta}) + \frac{\theta_n - b_n}{s_{renice} c_m} + \sum_{k=1}^{N} T_{\Sigma,k}((b_n, \theta_{-n}), (b_n, \theta_{-n})) &\geq 0. \end{aligned}$$

Using lemma 5.2, we get that it is enough for s_{renice} to satisfy

$$\frac{\theta_n - b_n}{s_{renice} c_m} - f_n^A \frac{\theta_n - b_n}{c_m} \geq 0.$$

Hence, it is sufficient that s_{renice} satisfies

$$s_{renice} \leq \frac{1}{f_n^A}. \tag{6.9}$$

The institution can declare a value for s_{renice} for every job, or take s_{renice} as

$$\min_{n \in \mathcal{N}} \frac{1}{f_n^A}. \tag{6.10}$$

■

Theorem 6.6 *In a system with RN, CLOSE and EARLY, the extended VCG mechanism can implement any social welfare function g in Ex-Post equilibrium.*

Proof: Follows immediately from lemma 6.4 and lemma 6.5. ■

6.4 An Implementation of a General Social Welfare Function in Dominant Strategies

Lemma 6.7 *Under the EVCG mechanism, in a system with $g \in \mathcal{G}$, POST, CLOSE and EARLY, an agent cannot gain by declaring less than her real type.*

Proof: $\forall \vec{\theta}, \vec{b} \in \Theta^N$, s.t. $b_n < \theta_n$, we wish to prove that

$$U_n((\theta_n, b_{-n}), \vec{\theta}) - U_n(\vec{b}, \vec{\theta}) \geq 0.$$

According to equation (6.1), the definition of the EVCG payment:

$$\begin{aligned} & U_n((\theta_n, b_{-n}), \vec{\theta}) - U_n(\vec{b}, \vec{\theta}) = \\ & \left\{-T_n((\theta_n, b_{-n}), \vec{\theta}) + T_n((\theta_n, b_{-n}), (\theta_n, b_{-n}))\right\} - \sum_{k \in \mathcal{N}} T_{\Sigma,k}((\theta_n, b_{-n}), (\theta_n, b_{-n})) \\ & + \left\{T_n(\vec{b}, \vec{\theta}) - T_n(\vec{b}, \vec{b})\right\} + \sum_{k \in \mathcal{N}} T_{\Sigma,k}(\vec{b}, \vec{b}). \end{aligned} \tag{6.11}$$

Due to the operation of $POST_n$, when agent n is lying:

$$\begin{aligned} T_n(\vec{b}, \vec{\theta}) = T_n(\vec{b}, \vec{b}) + \frac{\theta_n - b_n}{c_{m_1}} + s_{post} \quad + \quad D_n^A(\vec{b}, \theta_{-n}) \geq \\ T_n(\vec{b}, \vec{b}) + \frac{\theta_n - b_n}{c_{m_1}} + s_{post} \quad . \end{aligned}$$

where $n \in \mathcal{N}_{m_1}^A,\ A = o(\vec{b}, \vec{b})$. Hence,

$$\left\{T_n(\vec{b}, \vec{\theta}) - T_n(\vec{b}, \vec{b})\right\} \geq \frac{\theta_n - b_n}{c_{m_1}} + s_{post}. \tag{6.12}$$

When agent n is telling the truth, in this system, her output time may only be decreased (compared to her static output time), due to the revelation of other agents' lies:

$$\begin{aligned} & -T_n((\theta_n, b_{-n}), \vec{\theta}) = \\ & -T_n((\theta_n, b_{-n}), (\theta_n, b_{-n})) - D_n^{A''}(\theta_n, b_{-n}), \theta_{-n}) \geq \\ & -T_n((\theta_n, b_{-n}), (\theta_n, b_{-n})) \end{aligned}$$

where $A'' = o((\theta_n, b_{-n}), (\theta_n, b_{-n}))$. Hence,

$$\left\{-T_n((\theta_n, b_{-n}), \vec{\theta}) + T_n((\theta_n, b_{-n}), (\theta_n, b_{-n}))\right\} \geq 0. \tag{6.13}$$

Substituting equation (6.12) and equation (6.13) back in equation (6.11) we get

$$U_n((\theta_n, b_{-n}), \vec{\theta}) - U_n(\vec{b}, \vec{\theta}) \geq$$
$$-\sum_{k \in \mathcal{N}} T_{\Sigma,k}((\theta_n, b_{-n}), (\theta_n, b_{-n})) + \frac{\theta_n - b_n}{c_{m_1}} + s_{post} + \sum_{k \in \mathcal{N}} T_{\Sigma,k}(\vec{b}, \vec{b}).$$

Using lemma 5.2

$$U_n((\theta_n, b_{-n}), \vec{\theta}) - U_n(\vec{b}, \vec{\theta}) \geq$$
$$\frac{\theta_n - b_n}{c_m} - \frac{\theta_n - b_n}{c_{m'}} f_n^{A'} + s_{post},$$

where $A' = o_{\sum}((\theta_n, b_{-n}), (\theta_n, b_{-n}))$ and $n \in \mathcal{N}_{m'}^{A'}$.

So it suffices that s_{post} fulfills:

$$s_{post} \geq (\theta_n - b_n)\left(\frac{N}{\min_{j \in \mathcal{M}} c_j} - \frac{1}{c_m}\right) \geq (\theta_n - b_n)\left(\frac{f_n^{A'}}{c_{m'}} - \frac{1}{c_m}.\right)$$

That can be accomplished using the conditions imposed in the proof of lemma 5.8. ■

Theorem 6.8 *In a system with POST, CLOSE and EARLY, it is possible to implement a general social welfare function g in dominant strategies.*

Proof: Follows immediately from lemma 6.4 and lemma 6.7. ■

Chapter 7

Mechanism Qualities: Discussion

In this chapter we evaluate the EVCG mechanisms we suggested, mainly according to the properties suggested in chapter 4.

7.1 Limitations on Input

The RN based mechanism can deal with any set of $\vec{b}, \vec{\theta}$, and yet complete the execution of all jobs. The $POST$ based mechanism, on the other hand, must stop executing certain jobs, according to either of the following two criteria:

1. $\theta_n > H_{abs}$: if at time t the job has performed a total work of $\int_0^t X_n(t')dt' > H_{abs}$, the execution of the job is stopped.

2. $(\theta_n - b_n) > H_{diff}$: if at time t the job has performed a total work of $\int_0^t X_n(t')dt' > b_n + H_{diff}$, the execution of the job is stopped.

Using the first criterion will prevent the mechanism from performing certain long jobs at all. Obviously, jobs with declarations larger than H_{abs} cannot be submitted and guaranteed to be executed fully. By determining $H_{abs} > \max_{n \in \mathcal{N}} b_n$ in the realization stage, it can be verified that all the truth-telling jobs can be completed. On the other hand, setting a large value of H_{abs} means that s_{post} is increased accordingly, which in turn causes both the agents utility and the final social welfare to decrease, in case agents lie downward.

Using the second criterion, every job declaration is admissible for every value of H_{diff}, which therefore can be determined in advance. Using the H_{diff} criterion gains scalability.

7.2 Budget Considerations

The rent the institution gets depends on the implemented social welfare function. If $g_{\sum}$ is implemented, the rent is of course non-negative, since all payments are non-negative. Some social welfare functions may bring a negative rent though. For example,

- $g = -\sum_{k\in\mathcal{N}} (T_k - T_0)^2$.
- $\vec{c} = (1, 100)$.
- $T_0 = 100$.
- $\vec{b} = (100)$.

Under an optimal allocation A, in which $\mathcal{N}_1^A = \{1\}$,

$$X_1^A(t) = \begin{cases} 1 & 0 \le t < 100 \\ 0 & 100 \le t < \infty. \end{cases}$$

A is efficient, though not Work-Function-Pareto efficient. The rent is negative:

$$P_1(\vec{b}) = -T_1(\vec{b}, \vec{b}) + T_{\sum,1}(\vec{b}, \vec{b}) = -100 + 1 = -99 < 0.$$

As we can see, the rent can be made as low as desired, by setting the desired ending time T_0 to an appropriate value. Note that a function which is optimized by a Work-Function-Pareto non-efficient allocation, is not a regular social welfare function.

Theorem 7.1 *If the static allocation under the EVCG mechanism is Work-Function-Pareto efficient, the rent is positive.*

Proof: Under the EVCG mechanism, which allocates $\vec{b}$ under the static allocation A, the institution receives a rent of

$$\sum_{k\in\mathcal{N}} \left\{ -E_k^A + \sum_{n\in\mathcal{N}} E_n^{A_\Sigma} \right\} = -\sum_{k\in\mathcal{N}} E_k^A + N \sum_{n\in\mathcal{N}} E_n^{A_\Sigma},$$

where $A_\Sigma = o_{\sum}(\vec{b}, \vec{b})$.

We need to show that if A is Work-Function-Pareto-efficient, then

$$\sum_{k\in\mathcal{N}} E_k^A \leq N \sum_{n\in\mathcal{N}} E_n^{A_\Sigma}. \tag{7.1}$$

Let $c_{m'} := \max_{m\in\mathcal{M}} c_m$. Then for allocation A_Σ, let us arrange the jobs on each CPU according to length, shortest first:

$$\begin{aligned} N_m^{A_\Sigma} &:= \left|\mathcal{N}_m^{A_\Sigma}\right| \\ b_1^{m,A_\Sigma} \leq b_2^{m,A_\Sigma} \dots \leq\leq b_{N_m^{A_\Sigma}}^{m,A_\Sigma} & \\ N\sum_{k\in\mathcal{N}} E_k^{A_\Sigma} = N\sum_{m\in\mathcal{M}} \frac{1}{c_m} \sum_{i=1}^{N_m^{A_\Sigma}} \left(N_m^{A_\Sigma} - i + 1\right) b_i^{m,A_\Sigma} &\geq \\ N\sum_{m\in\mathcal{M}} \frac{1}{c_m} \sum_{i=1}^{N_m^{A_\Sigma}} b_i^{m,A_\Sigma} &\geq N\frac{1}{c_{m'}} \sum_{i=1}^{N} b_i \end{aligned}$$

where $\vec{b}^{m,A_\Sigma}$ is the vector of declared lengths of the jobs in $\mathcal{N}_m^{A_\Sigma}$, sorted according to their order on CPU m, under allocation A_Σ.

On the other hand, for any allocation which is Work-Function-Pareto efficient, an agent cannot be better off by executing her job earlier on the same CPU (according to corollary 3.3.1), neither can she be better off by using another CPU, after its share of the work is done. Let $E^{m,A} := \max_{k\in\mathcal{N}_m^A} E_k^A$ denote the **Ending time of CPU** m. For a Work-Function-Pareto-efficient allocation A, $\forall m \in \mathcal{M}$,

$$\begin{aligned} E^{m,A} &= \frac{\sum_{i\in\mathcal{N}_m^A} b_i}{c_m} \\ \forall k \in \mathcal{N}\ \ E_k^A &\leq E^{m,A} + \frac{b_k}{c_m} \end{aligned} \tag{7.2}$$

Hence, comparing the ending time of jobs to the ending time of CPU m', we can divide the jobs into two sets:

- $\mathcal{I} := \{k | E_k \leq E^{m',A}\}$
- and $\mathcal{J} := \{k | E^{m',A} < E_k \leq E^{m',A} + \frac{b_k}{c_{m'}}\}$.

Summing over those sets, we get

$$\sum_{k\in\mathcal{N}} E_k^A \leq N E^{m',A} + \frac{1}{c_{m'}} \sum_{k\in\mathcal{J}} b_k$$

Due to equation (7.2), and since $\mathcal{J} \subseteq \mathcal{N} \backslash \mathcal{N}^A_{m'}$

$$NE^{m',A} + \frac{1}{c_{m'}} \sum_{k \in \mathcal{J}} b_k =$$

$$N \frac{1}{c_{m'}} \left(\sum_{k \in \mathcal{N}} b_k - \sum_{k \notin \mathcal{N}^A_{m'}} b_k \right) + \frac{1}{c_{m'}} \sum_{k \in \mathcal{J}} b_k \leq$$

$$\frac{1}{c_{m'}} \left(N \sum_{k \in \mathcal{N}} b_k + (1-N) \sum_{k \in \mathcal{J}} b_k - N \sum_{k \in \mathcal{N} \backslash \mathcal{N}^A_{m'} \backslash \mathcal{J}} b_k \right) \leq \frac{1}{c_{m'}} N \sum_{k \in \mathcal{N}} b_k.$$

Hence, equation (7.1) holds:

$$\sum_{k \in \mathcal{N}} E^A_k \leq \frac{1}{c_{m'}} N \sum_{k \in \mathcal{N}} b_k \leq N \sum_{k \in \mathcal{N}} E^{A_\Sigma}_k$$

and the rent cannot be negative.

■

Corollary 7.2.1 *In an EVCG mechanism which implements a regular social welfare function, the rent is non-negative.*

Proof: Follows immediately from theorem 7.1 and from corollary 3.3.2. ■

7.3 Safety Margins

A *POST-EARLY-CLOSE* mechanism only has upper safety margins. An *RN-EARLY-CLOSE* mechanism, on the other hand, does have safety margins.

7.4 Justness

The $POST-EARLY-CLOSE$ mechanism is just, while the $RN-EARLY-CLOSE$ mechanism is not. Making sure liars do not benefit from other agents' lies, makes the mechanism able of implementing a social welfare function in Ex-Post Equilibrium. Justness is what makes the mechanism able of implementing a social welfare function in dominant strategies.

7.5 Individual Rationality

So far, we have ignored the agent's valuation of the execution of her job, V_n in equation (3.2), since we assumed an agent who submitted a declaration cannot decide to withdraw from the game.

Now let us consider allowing agents to withdraw after the realization stage. A rational agent will choose to withdraw if $V_n < T_n + P_n$. The valuation V_n is the agent's secret. It does not depend on the rules of the game, nor on the other agents. This valuation may depend though on an alternative CPU which the agent has at her disposal (e.g. a slow, private, free of charge desktop).

As we mentioned in chapter 3, footnote 2, it is more common in the literature to assume the agent's secret is her valuation of the allocation rather than the length of her job. Taking the valuation into account as a secret brings us back to the common notion of secret: the agent's valuation of a good.

We propose a mechanism built on top of the proposed EVCG mechanism, using two rounds: After stage 3, the declaration stage, agents may withdraw without paying. They do not get their jobs done, of course. In the next round, the agents are not allowed to change their declarations: the allocation is calculated based on the previous declarations of those who stayed. The payment for the staying agents is based on the staying agents only. Let "$\hat{}$" denote functions and allocations on the second round. Then,

$$\hat{P}_n(\vec{b}) = -\hat{T}_n(\vec{b},\vec{b}) + \sum_{k\in\hat{\mathcal{N}}} \hat{T}_{\Sigma,k}(\vec{b},\vec{b}).$$

We wish to show that for the two-rounds EVCG mechanism:

- Indeed two rounds are enough: a rational agent who stayed for the second round, would get a positive utility.
- Truth telling is in ex-post equilibrium.

Lemma 7.2 *In the EVCG mechanism of two rounds, if agent n stayed for the second round, she will have a positive utility, assuming everyone else is truth telling.*

Proof: For truth telling agents, when all other agents are truth telling, every agent's utility equals $g_{\sum}$. The set of jobs to be executed only decreased after withdrawal, so

$$g_{\sum}(\vec{E}^{\hat{A}}) \geq g_{\sum}(\vec{E}^{A})$$

where $A = o(\vec{b}, \vec{\theta})$ is the original allocation, and $\hat{A} = \hat{o}(\vec{b}, \vec{\theta})$ is the allocation formed for the agents who chose to stay.

Hence, every truth telling agent who stayed can only be better off, now that some agents have retired. If participating was individually rational before,

$$U_n = V_n - T_n(\vec{b}, \vec{b}) - P_n(\vec{b}) > 0$$

it is all the more individually rational now,

$$\begin{aligned} \hat{U}_n(\vec{b}, \vec{b}) = V_n - \hat{T}_n(\vec{b}, \vec{b}) - \hat{P}_n(\vec{b}) &= \\ V_n + g_{\sum}(\vec{E}^{\hat{A}}) &\geq \\ V_n + g_{\sum}(\vec{E}^{A}) &= U_n(\vec{b}, \vec{b}) > 0 \end{aligned} \tag{7.3}$$

Hence, there is no agent who stayed to the second round, but wishes to quit then: the maximal number of needed rounds is two. ■

What about agents who withdrew? Could it be that it would be worthwhile for an agent to lie, thus making it non-beneficial for another agent to stay, and improving her own utility by that?

Example 7.5.1 *In an EVCG mechanism with either RN or POST, CLOSE and EARLY, it may be worthwhile for an agent to lie, thus making it non-beneficial for another agent to stay, and improving her own utility by that.*

Proof: Let $\vec{c} = (1), \vec{\theta} = (1,1), g = g_{\sum}$. Assume agent 2 tells the truth. If agent 1 also tells the truth, $U_1(\vec{\theta}, \vec{\theta}) - V_1 = U_2(\vec{\theta}, \vec{\theta}) - V_2 = -1 - 2 = -3$. If agent 1 lies and declares $b_1 = 2$, $U_2((2,1), \vec{\theta}) = V_2 - 1 - 3 = V_2 - 4$. If $4 > V_2 > 3$, agent 2 would withdraw. Then agent 1's utility would be $U_1((2),(1)) = V_1 - 2$, which is better than her utility from telling the truth. ■

To prevent this situation, a certain change in the output time of lying agents is required, if there is a second round.

Lemma 7.3 *There exists a two-round mechanism which allows withdrawing, for the RN based EVCG mechanism, which is IC in ex-post equilibrium.*

Proof: Assume $b_{-n} = \theta_{-n}$. In order to make lying non-beneficial for agent n, we require:

$$\hat{U}_n((b_n, \theta_{-n}), \vec{\theta}) \leq \hat{U}_n(\vec{\theta}, \vec{\theta}).$$

Note that the subset $\hat{\mathcal{N}}$ depends on the declarations; hence different agents may decide to withdraw, depending on agent n's declaration.

A liar who withdrew is not better off than a truth teller who withdrew. Both their utilities are zero. Let us concentrate then on agent n who lies and stays.

From equation (7.3) we know that when all agents are truth telling, if agent n stayed, then

$$U_n(\vec{\theta},\vec{\theta}) \leq \hat{U}_n(\vec{\theta},\vec{\theta}).$$

It will suffice then to construct the mechanism such that

$$\hat{U}_n((b_n,\theta_{-n}),\vec{\theta}) \leq U_n(\vec{\theta},\vec{\theta})$$

which leads to the demand

$$-\hat{T}_n((b_n,\theta_{-n}),\vec{\theta}) + \hat{T}_n((b_n,\theta_{-n}),(b_n,\theta_{-n})) - \sum_{k\in\hat{\mathcal{N}}} T_{\Sigma,k}((b_n,\theta_{-n}),(b_n,\theta_{-n})) \leq \\ - \sum_{k\in\mathcal{N}} T_{\Sigma,k}(\vec{\theta},\vec{\theta})$$

In order to fulfill this demand, a lying agent must receive her output on the second round s.t. it fulfills the following demand:

$$\hat{T}_n((b_n,\theta_{-n}),\vec{\theta}) - \hat{T}_n((b_n,\theta_{-n}),(b_n,\theta_{-n})) \geq \\ \sum_{k\in\mathcal{N}} T_{\Sigma,k}(\vec{\theta},\vec{\theta}) - \sum_{k\in\hat{\mathcal{N}}} T_{\Sigma,k}((b_n,\theta_{-n}),(b_n,\theta_{-n})) \tag{7.4}$$

In other words, the output time for a lying agent must be later than the output time for a truth telling agent (of the same declaration) at least by the gain to the $g_{\sum}$ social welfare function of the declarations due to the second round.

In order to achieve that, the two-round mechanism can use, for example,

$$POST_n(\hat{T}_n(\vec{b},\vec{\theta}), \\ \hat{T}_n(\vec{b},\vec{b}) + \sum_{k\in\mathcal{N}} T_{\Sigma,k}((\theta_n,b_{-n}),(\theta_n,b_{-n})) - \sum_{k\in\hat{\mathcal{N}}} T_{\Sigma,k}(\vec{b},\vec{b}), \\ \theta_n + \epsilon) \tag{7.5}$$

for some positive value of ϵ. Note that this $POST$ operator is activated on top of the regular $POST$ operator which the EVCG mechanism uses: it is activated at time $\hat{T}_n(\vec{b},\vec{\theta})$, when the agent was

supposed to get her output, according to the EVCG mechanism. Then another delay is added, to insure that when other agents are truth telling, equation (7.4) is fulfilled. ■

Lemma 7.4 *There exists a two-round mechanism (which allows withdrawing), for the POST based EVCG mechanism, which is IC in dominant strategies.*

Proof: In order to make truth telling a dominant strategy for agent n, we require:

$$\hat{U}_n((b_n, b_{-n}), \vec{\theta}) \leq \hat{U}_n((\theta_n, b_{-n}), \vec{\theta}).$$

We concentrate again on an agent n who lies and stays.

From equation (7.3) and from the justness of the $POST$ based mechanism,

$$U_n((\theta_n, b_{-n}), (\theta_n, b_{-n})) \leq \hat{U}_n((\theta_n, b_{-n}), (\theta_n, b_{-n})) \leq \hat{U}_n((\theta_n, b_{-n}), \vec{\theta}).$$

It will suffice then to construct the mechanism such that

$$\hat{U}_n((b_n, b_{-n}), \vec{\theta}) \leq U_n((\theta_n, b_{-n}), (\theta_n, b_{-n})).$$

which leads to the following demand on n's output time in the second round:

$$\hat{T}_n(\vec{b}, \vec{\theta}) - \hat{T}_n(\vec{b}, \vec{b}) \geq$$
$$\sum_{k \in \mathcal{N}} T_{\Sigma,k}((\theta_n, b_{-n}), (\theta_n, b_{-n})) - \sum_{k \in \hat{\mathcal{N}}} T_{\Sigma,k}(\vec{b}, \vec{b})$$

This demand will be satisfied by activating $POST$ according to equation (7.5). ■

Note 7.5.2 *The two-round mechanisms are hardly practical. The amount of time by which the output is postponed depends on an optimization done on (θ_n, b_{-n}). Hence, while the two-round mechanisms are IC, they differ from the single-round EVCG mechanism by requiring heavy computations to be done whenever an agent lies. Their low computability makes them less practical.*

7.6 Final Social Welfare

When comparing mechanisms on all the domain Θ^N, some mechanisms are strictly better than others.

- An EVCG mechanism which uses the $CLOSE$ and $EARLY$ operators has a better final social welfare than an EVCG mechanism which does not.

- When we proved that the EVCG mechanism can implement a social welfare function, we actually defined a range for the parameters s_{post} and s_{renice}. When the parameters are within that range, the function can be implemented. An EVCG mechanism which uses the threshold values for s_{post} or for s_{renice} has a better final social welfare than an EVCG mechanism which chooses other values, which are strictly within the allowed range (a lower value for s_{renice} or a higher value for s_{post}). The higher s_{renice} is, the faster downward liars' jobs are terminated, and regular execution continues. The lower s_{post} is, the sooner the execution of downward liars' jobs is resumed.

- An EVCG mechanism which sets a different value of $s_{post,n}$ for each agent according to equation (5.12), taking c_m into account, has a better final social welfare than a mechanism which uses one common value for all the agents, according to $\max_{n \in \mathcal{N}} s_{post,n}$.

- An EVCG mechanism which sets s_{renice} individually for every job, according to equation (6.9) has a better final social welfare than an EVCG mechanism which uses equation (6.10) to set s_{renice}.

Setting an optimal value for H_{abs} and H_{diff}, on the other hand, is not that straightforward, but involves a trade-off. Lowering those values enables setting a lower value for s_{post}, which makes the final social welfare better. At the same time, lowering those threshold values means that the range $\hat{\Theta}$ on which the jobs get to terminate at all becomes smaller. Achieving a better final social welfare for small lies comes at the expense of not being able to complete jobs of larger lies, as discussed in section 7.1.

7.7 Complexity and Off-Line Calculations

The problem of calculating the best schedule is NP-hard. The time complexity of a naive algorithm for calculating the best schedule, given the lengths of the jobs, is $O((N+M)!)$.

We have proved here that the EVCG mechanism implements a general social welfare function, under the assumption that the institution is indeed able to calculate (exactly) the optimal schedule.

It may seem that due to the computability issue, using the EVCG mechanism is not worthwhile, but there are several scenarios in which it is practical:

1. When the typical job length is long, relative to the time it takes to calculate the allocation. This time depends N and M, and not on $\vec{\theta}$:

$$(N+M)!t_c << \frac{\sum \theta_k}{\sum c_m}$$

   where t_c is the time to calculate the social welfare function over one allocation, and is $O(N^2)$ (using a naive algorithm). The time to find the optimal allocation would also decrease drastically if the cluster was composed of identical machines $c_1 = \ldots = c_M$, or of subsets of identical machines. Subsets of identical declarations $\vec{b}$ would also make the task of finding an optimal allocation faster to compute.

2. When the variance of $\vec{\theta}$ is high. Then $\max_{FUL(\vec{\theta})} g(\vec{T}(\vec{\theta},\vec{\theta})) - \min_{FUL(\vec{\theta})} g(\vec{T}(\vec{\theta},\vec{\theta}))$ is relatively high, and the institution can gain a lot from a truth revealing mechanism.

3. When $\Theta \subset \mathbb{N}_+$, in a repeated game, it is possible to remember the optimal allocations for certain sets of declarations.[1]

In all those cases, the calculations needed to find the optimal allocation end before the beginning of the execution (off-line scheduling). The job control tools do not require any calculations. Thus, even if agents lie, the dynamic allocation is known at once: the cluster needs not halt and wait for decisions when a job control operator is activated.

7.8 Practical Limitations on the Implementation

When coming to implement the mechanism we suggest in this work, one may encounter several "real world" problems:

1. The mechanism requires monetary transfers. In order to implement that, the agents are required to be economic units. If they work for the institution which owns the cluster, this may not be the case.

[1] Thus replacing the complexity of calculating the optimal schedule with memory complexity.

2. The mechanism we suggest counts on the institution's ability to enforce the optimal schedule. In real life, this ability may be limited, due to the agents having direct access to the cluster. If the agents have direct access to the cluster, the strategy of reporting the job's true length and submitting it to the institution for allocation may be inferior to submitting the job directly to a CPU of the agent's choice.

Chapter 8

Summary

In this work, we presented the problem of Off-Line Cluster Scheduling, where the institution has the ability to set the allocation of jobs to CPUs, but it does not know the jobs' lengths. The institution also has the ability to affect the allocation in a limited manner after the execution has begun, using job control tools. The agents, each owning one job, have a certain utility from the completion of the execution of jobs, as well as from monetary transfers. The institution has a utility from the completion of the jobs, which is a social welfare function.

We devised a mechanism, in which the agents' best strategy is telling the institution what the real length of their job is. We formulated two variants of this mechanism:

1. A mechanism which uses the job control tool $POST$, and implements a general social welfare function in dominant strategies. This variant is just, but it only has upper safety margins, and it poses limitations on the input.

2. A mechanism which uses the job control tool RN, and implements a general social welfare function in Ex-Post equilibrium. This mechanism has safety margins, and it does not pose any limitation on the input, but it is not just. This mechanism is more suitable to a situation where the agent herself does not know exactly the length of her job, but instead she has distributional information over it.

We also proved that both monetary payments and verification using job control tools are essential in order to implement a social welfare function. A mechanism similar to what we suggested, only

without monetary transfers, will not implement even the "sum of utilities" function. A similar mechanism which uses monetary transfers, but does not use the job control tools in order to verify that the agents are truth telling and punish them otherwise, will not implement that function either.

The mechanisms we suggested are not IR. There exists another mechanism in which agents can withdraw before paying, which implements a general social welfare function. In a variant of the mechanisms we suggested, we enable the agents to withdraw from the game once they know the price they will have to pay. This variant is individually rational, and still implements a general social welfare function with the same level of incentive compatibility, as the original variant it sprung from (either RN or $POST$ based). This mechanism it includes on-line calculations of a schedule when agents lie, though, in order to determine the severity of the punishment.

The rent in those mechanisms is non-negative if the static allocation is Work-Function-Pareto-efficient.

In this work we proved that it is possible to implement a general social welfare function in the off-line cluster scheduling environment, by supplying a family of such mechanisms.

References

[1] A. H. Abdekhodaee and A. Wirth. Scheduling parallel machines with a single server: some solvable cases and heuristics. *Computers and Operations Research*, 29 (3), 2002.

[2] A. V. Ackere. Conflicting interests in the timing of jobs. *Management Science*, 36(8), 1990.

[3] A. V. Ackere. The impact of conflicting interests on the choice of an appointments system. *Belgian Journal of Operations Research, Statistics and Computer Science*, 31 (3-4), 1992.

[4] E. Altman and N. Shimkin. Individual equilibrium and learning in process sharing systems. Technical Report CC212, Department of Electrical Engineering, Technion, October 1997.

[5] Y. Azar and L. Epstein. On-line scheduling with precedence constraints. In *Proc. of 7th SWAT*, pages 164–174, 2000.

[6] M. Azizoglu and O. Kirca. Tardiness minimization on parallel machines. *International Journal of Production Economics*, 55 (2):163–168, 1998.

[7] A. Barak, S. Guday, and R. Wheeler. *The MOSIX Distributed Operating System, Load Balancing for UNIX*, volume 672. Springer-Verlag, 1993. http://www.mosix.org.

[8] A. Barak, O. La'adan, and A. Shiloh. Scalable cluster computing with MOSIX for LINUX. In *Proc. Linux Expo '99*, pages 95–100. Raleigh, N.C., May 1999. http://www.mosix.org.

[9] R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In *The 4th International*

Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000). IEEE Computer Society Press, USA, May 2000.

[10] E. Clarke. Multipart pricing of public goods. *Public Choice*, 18:19–33, 1971.

[11] L. Cons, K. Berry, O. Bachmann, et al. Bash reference manual: Job control. http://www.gnu.org/manual/bash-2.05a/html_chapter/bashref_7.html.

[12] B. Dodin and A. Elimam. Integrated project scheduling and material planning with variable activity duration and rewards. *IIE Transactions*, 33 (11):1005–1018, Nov. 2001.

[13] J. Du and J. Y.-T. Leung. Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15:483–494, 1990.

[14] B. Falsafi and M. Lauria, editors. *REXEC: A Decentralized, Secure Remote Execution Environment for Clusters.*, volume 1797 of *Lecture Notes in Computer Science*. Springer, 2000.

[15] D. G. Feitelson and A. M. Weil. Utilization and predictability in scheduling the IBM SP2 with backfilling. In *12th Intl. Parallel Processing Symp.*, pages 542–546, 1998.

[16] T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

[17] J. C. Harsanyi. Games with incomplete information played by 'Bayesian' players. Parts i-iii. *Management Science*, 14:159–182, 320–334, 486–502, 1967-8.

[18] R. Holzman, N. Kfir-Dahav, D. Monderer, and M. Tennenholtz. Bundling equilibrium in combinatorial auctions. mimeo, Technion, http://iew3.technion.ac.il/ moshet/rndm11.ps, 2001.

[19] L. Hurwicz. On allocations attainable through Nash equilibria. *Journal of Economic Theory*, 21:140–165, 1979.

[20] U. Kjems. Jobd. http://bond.imm.dtu.dk/jobd/.

[21] D. A. Lifka. The ANL/IBM SP scheduling system. In D. G. Feitelson and L. Rudolph, editors, *IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 295–303. Springer, Berlin Lecture Notes in Computer Science LNCS 949, 1995.

[22] M. Livny et al. Condor - high throughput computing. http://www.cs.wisc.edu/condor/.

[23] A. Mas-Colell, M. Whinston, and J. Green. *Microeconomic Theory*. Oxford University Press, 1995. Chapter 23: Incentives and Mechanism Design.

[24] J. A. V. Mieghem. Dynamic scheduling with convex delay costs: The generalized cμ rule. *The Annals of Applied Probability*, 5(3):808–833, 1995.

[25] K. Mount and S. Reiter. The informational size of message spaces. *Journal of Economic Theory*, 8:161–191, 1974.

[26] R. B. Myerson. Incentive compatibility and the bargaining problem. *Econometrica*, 47 (1):61–73, 1979.

[27] P. Naor. The regulation of queue size by levyying tolls. *Econometrica*, 37:15–24, 1969.

[28] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001.

[29] D. Nowak. ASCI at Livermore- Alliances. http://www.llnl.gov/asci/alliances/. U.S. Department of Energy under Contract W-7405-Eng-48.

[30] J. Root. Scheduling with deadlines and loss functions on k parallel machines. *Management Science*, 14:460–475, 1965.

[31] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: An economy driven job scheduling system for clusters. Technical report, The University of Melbourne, July 2002. http://www.cs.mu.oz.au/~raj/grids/papers/libra.pdf.

[32] J. A. Stankovic and I. S. Sidhu. An adaptive bidding algorithm for processes, clusters and distributed groups. In *Proc. of 4-th International Conf. on Distributed Computing Systems*, pages 49–59. IEEE Computer Society, 1984.

[33] W. R. Stevens. *Advanced Programming in the UNIX Environment*. Addison Wesley Longman, Inc, 1993.

[34] A. S. Tanenbaum. *Operating Systems: Design and Implementation*. Prentice Hall, Inc. Englewood Cliffs, New Jersey 07632, 1987.

[35] L. Torvalds et al. Debian GNU/linux - the universal operating system. http://www.debian.org/.

[36] L. Torvalds et al. Linux kernel. http://www.kernel.org/.

[37] W. Vickrey. Counterspeculations, auctions, and competetive sealed tenders. *Journal of Finance*, 16:15–27, 1961.

[38] M. P. Wellman, W. E. Walsh, P. R. Wurman, and J. K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35(1):271–303, 2001.

[39] D. Wiltzius. Proposed 2000 and 2001 Livermore computing services to ASCI Alliance sites. http://www.llnl.gov/asci/alliances/services_to_alliances.html, 2000. U.S. Department of Energy under Contract W-7405-Eng-48.

תמריצים ופונקציות רווחה כלליות בבעיית הזימון הבלתי מקוון של אשכול מחשבים

אורנה אגמון

תמריצים ופונקציות רווחה כלליות בבעיית הזימון הבלתי מקוון של אשכול מחשבים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת תואר
מגיסטר למדעים
במתמטיקה שמושית

אורנה אגמון

הוגש לסנט הטכניון — מכון טכנולוגי לישראל

אדר א' תשס"ג חיפה פברואר 2003

המחקר נעשה בהדרכת ד"ר רן סמורודינסקי
במסגרת התוכנית הבין-יחידתית למתמטיקה שמושית

הכרת תודה

תודה בעברית

הקדשה בעברית

תוכן עניינים

תקציר

לכל אחת ממספר שחקניות יש עבודת חישוב אחת לבצע. לצורך ביצוע העבודה עומד לרשותן משאב חישוב המורכב ממספר מעבדים בעלי עוצמות שונות, המנוהלים באופן ריכוזי- אשכול מחשבים. האשכול נמצא בבעלותו של מוסד מסוים. כמות העבודה שיש לכל שחקנית לבצע היא מידע פרטי של אותה שחקנית. התועלת של שחקנית יורדת באופן לינארי ככל שזמן קבלת הפלט מעבודת החישוב שלה עולה.

המוסד רוצה למרב את תועלתו הוא: פונקציית רווחה חברתית כללית של תועלות השחקניות. הוא רוצה לעשות כן על ידי זימון העבודות על המחשבים. למשל, המוסד עשוי לרצות למזער את סכום זמני קבלת הפלט, את זמן קבלת הפלט המאוחר ביותר (מנעד זמני קבלת הפלט), או כל פונקציה שהיא של זמני קבלת הפלט של השחקניות והתשלומים שהן משלמות. הבעיה היא שכדי לפעול באורח מיטבי, על המוסד לדעת את אורכן האמתי של עבודות החישוב- מידע שאין ברשותו.

אנו מציעים משפחה של מנגנונים אשר בהם השחקניות מעדיפות להצהיר על אורכן האמתי של עבודות החישוב שלהן, בעוד שהמוסד משתמש בהצהרות אלו כדי לזמן הקצאה של זמן מחשב לעבודות השונות כך שהוא ממרב פונקציית רווחה חברתית כלשהי. מנגנונים אלו כרוכים הן בתשלומים בטרם ביצוע העבודות והן ביישומם של כלי שליטה בעבודות במהלך הביצוע. כלי שליטה בעבודות הוא שיטה מסוימת לשינוייה של הקצאה התחלתית או של זמני הפלט הצפויים. דוגמאות לכלי שליטה המשנים את ההקצאה הן: הפסקת ביצועה של עבודה מסוימת, דחייה של המשך עבודה מסוימת לזמן מאוחר יותר ("דחייה"), הקטנת חלקו של המעבד המופנה לצרכי עבודה מסוימת, כלומר הקטנת יעילות המעבד ("הקטנה"), או סגירת פערים בניצולו של המעבד על ידי ביצוע מוקדם של עבודות שנועדו לזמן מאוחר יותר ("סגירה"). מתן הפלט בזמן מוקדם יותר מן המתוכנן, אך כמובן שלא מוקדם יותר מזמן הסיום של העבודה ("הקדמה"), הוא דוגמה לכלי שליטה המשנה את זמני קבלת הפלט. כלי השליטה עשויים להעדר ממערכת זו או אחרת. המשותף לכל כלי השליטה בעבודות הוא שאינם כרוכים בחישובים מורכבים (כדוגמת מציאת הקצאה מיטבית). עדיין, בשווי משקל אין צורך להשתמש בכלי שליטה אלו.

התשלום אותו אנו מציעים בנוי כהרחבה של תשלום "ויקרי קלארק גרובס" (וק"ג). תחת מנגנון וק"ג, אשר

בסביבות רבות מיישם את פונקציית סכום התועלות באסטרטגיות שולטות, ממרב המוסד את פונקציית סכום התועלות, וקובע את התשלום של שחקנית אחת על פי סכום התועלות של חברותיה. מכיוון שבסביבת הזימון הלא מקוון של אשכול המחשבים עשויה תועלתה של שחקנית אחת להיות תלויה באורכן האמתי של עבודותיהן של חברותיה, מידע פרטי אשר מתגלה רק במהלך ביצוע העבודות, יש הכרח בנקיטת פעולות בשלב זה. שני המנגנונים העיקריים שנתמקד בהם משתמשים האחד בהקטנת יעילות המעבד (מנגנון הקטנה), והשני בדחיית המשך ביצוע העבודה (מנגנון דחייה) כדי להשיג מטרה זו. כדי ליישם פונקציית רווחה כללית אין די בתשלומי וק"ג. לשם כך אנו מציעים לקבוע את התשלום שמשלמת שחקנית i באופן הבא:

$$-T_i + \sum_{k=1}^{N} T_{\Sigma,k}$$

כאשר T_i מייצג את זמן קבלת הפלט המחושב מראש בהתאם להצהרותיהן של כל השחקניות, כאשר המוסד בוחר את ההקצאה אשר ממרבת את פונקציית הרווחה שברצונו למרב, N מייצג את מספר השחקניות ואילו $T_{\Sigma,k}$ מייצג את הזמן המחושב מראש בו הייתה שחקנית k מקבלת את הפלט, לו עבור אותן הצהרות היה המוסד ממרב את פונקציית הרווחה $g_{\sum} = -\sum_{k=1}^{N} T_k$. אנו מכנים תשלום זה בשם תשלום "ויקרי קלארק גרובס מורחב" (וקג"מ). ניתן לראות את תשלום וקג"מ כאילו הוא מורכב מתמריץ ופיצוי. חלק התמריץ זהה לתשלום וק"ג המקורי. חלק הפיצוי מפצה על ההפרש בין זמן קבלת הפלט המחושב על פי הפונקציה שהמוסד אכן ממרב, לבין זמן קבלת הפלט המחושב לו היה המוסד ממרב את הפונקציה $g_{\sum}$.

את כלי השליטה בעבודות אנו מזניקים באופן הבא: אם מסתבר כי אורכה האמתי של עבודה קצר מאורכה המוצהר, מופעל הכלי "סגירה" כדי לאפשר לעבודות הממתינות לאותו מעבד להתבצע מוקדם יותר. אם מסתבר כי אורכה האמתי של עבודה ארוך יותר מאורכה המוצהר, מופעל אחד מן הכלים "דחייה" או "הקטנה", על פי הקיים במערכת, אך לא שניהם. אם מסתבר כי השחקנית הצהירה במדויק על אורכה של העבודה, מופעל הכלי "הקדמה", כך שאם עבודתה של שחקנית דוברת אמת החלה מוקדם מכפי שתוכנן, יכולה היא ליהנות מכך.

מנגנונים אלו נבחנים לאורם של מספר קריטריונים. ראש וראשון הוא עצמת תאימות התמריץ של המנגנון: האם אמירת אמת, בעוד שהמוסד מיישם פונקציית רווחה כללית, היא אסטרטגיה שולטת או שהיא רק בשווי משקל בדיעבד (אסטרטגיה נמצאת בשווי משקל בדיעבד אם כדאי לשחקנית לנקוט בה כנגד האסטרטגיות בהן נוקטות שאר השחקניות, ללא תלות באורכיהן של העבודות שלהן)? מנגנון הדחייה מיישם פונקציית רווחה כללית באסטרטגיות שולטות, בעוד שמנגנון ההקטנה מיישמה בשווי משקל בדיעבד.

איזון התקציב תלוי כמובן רק באופן קביעת התשלומים. עבור פונקציות רווחה רגולריות, כאלה שערכן גדל כאשר תועלתה של אחת השחקניות גדלה ואילו של האחרות נותרת קבועה, ניתן לקבוע כי אין על המוסד להשקיע כסף כדי לקיים את המנגנון.

מנגנון הדחייה הוא מנגנון הוגן: מובטח בו כי זמני קבלת הפלט לא יהיו מאוחרים יותר מן המשתמע מן ההקצאה ההתחלתית. מנגנון ההקטנה אינו כזה.

מנגנון ההקטנה מסתלם: אין הוא מציב כל מגבלה על כמות העבודה בה הוא מסוגל לטפל. מנגנון הדחייה לעומתו מחייב את המוסד לקבוע מדיניות לפיה שחקניות ששיקרו מעבר לסף מסוים (או שעבודתן ארוכה מסף אחר) חייבות להענש בכך שעבודותיהן לא יבוצעו כלל.

משמעותם של שולי בטחון של מנגנון היא רציפות פונקציית התועלת של שחקנית, עבור הכרזות קבועות, כאשר מסתבר כי אורכה האמתי של העבודה של אותה שחקנית שונה מהצהרתה. לשולי בטחון יש חשיבות מרובה כאשר השחקניות עצמן מחזיקות במידע בלתי מדויק אודות כמות העבודה הדרושה להן. למנגנון הדחייה שוליים מצד אחד מכיוון שתועלת השחקנית נפגעת באופן בלתי רציף כאשר אורכה של העבודה עולה על המוצהר. למנגנון ההקטנה לעומתו שולי בטחון משני העברים.

המחירים במשפחת מנגנונים זו ידועים מראש. עובדה זו מאפשרת לבנות על גבי המנגנונים המוצעים מנגנון חדש, דו-שלבי, אשר יאפשר לשחקניות לנשור מן המשחק בראותן כי התשלום הנדרש גבוה מדי. כדי שאמירת אמת תהיה אסטרטגיה שולטת במנגנון הדו-שלבי מבוסס ה"דחייה", ואסטרטגיה בשווי משקל בדיעבד במנגנון הדו-שלבי מבוסס ה"הקטנה", יש צורך בהפעלת עונש של דחיית זמן קבלת הפלט כאשר מתגלה שקר כלשהו במנגנון הדו-שלבי. מידת הדחייה אינה עוד מספר הנקבע מראש, אלא גורם התלוי בחישובים של בחירת הקצאה מיטבית בהסתמך על אורכה האמתי של העבודה של השחקנית ששיקרה. צורך זה מוציא את פעולת הענישה במנגנון הדו שלבי מגדר "כלי שליטה בעבודות".

מכיוון שפונקציית הרווחה אינה רק מדד לטיב בחירת המוסד, אלא גם פונקציית התועלת של המוסד עצמו, הרי שיש משמעות לערכה גם כאשר אחת או יותר מן השחקניות טועה (או משקרת) בהצהרתה. מנגנונים מסוימים בתוך המשפחה המוצעת טובים יותר מבחינה זו מאשר אחרים. למשל, מנגנונים העושים שימוש בכלים "סגירה" ו"הקדמה". מתוך טווח רציף של עצמת הכלי "הקטנה", עדיפים אלו אשר מקטינים פחות את עצמת המעבד. טווח עצמת הכלי "דחייה" קשור בקשר ישיר בסף השקר אשר מעבר לו מופסקת העבודה לחלוטין, או בסף כמות העבודה הנדרשת. עבור סף קבוע, עדיפים מנגנונים אשר דוחים את המשך העבודה לזמן מוקדם ככל האפשר.

אף אחד משני חלקי המנגנונים שאנו מציעים אינו מיותר. הפעלת כלי השליטה בעבודות באופן שהצענו ללא קבלת תשלומים יגרום לאמירת אמת להיות בלתי כדאית במצבים מסוימים, אף אם כל השחקניות האחרות דוברות אמת. העברת תשלומי וק"ג כאשר כלי שליטה אינם מופעלים כלל, אלא כל שנקבע מראש הוא חלוקת העבודות למעבדים השונים וסדר ביצוען על כל מעבד, תביא אף היא לאותן תוצאות.