Managing Shared Resources in the Data Center Era

by

Seyed Majid Zahedi

Department of Computer Science
Duke University

Date: ____________________
Approved:

____
Benjamin C. Lee, Supervisor

____
Vincent Conitzer

____
Jeffrey S. Chase

____
Kamesh Munagala

____
Carl A. Waldspurger

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2018

Abstract

Managing Shared Resources in the Data Center Era

by

Seyed Majid Zahedi

Department of Computer Science
Duke University

Date: ____________________________
Approved:

____
Benjamin C. Lee, Supervisor

____
Vincent Conitzer

____
Jeffrey S. Chase

____
Kamesh Munagala

____
Carl A. Waldspurger

An abstract of a dissertation submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2018

Abstract

To improve efficiency and amortize cost over more computation, resource sharing has become vital in high performance computing systems. In such systems, the conventional wisdom assumes that users have to share, regardless of the management policy. With a wide range of computing options available, this assumption does not seem to hold for today's self-interested users. These users selfishly pursue their individual performance without regard for others or the system. And if they dislike management outcomes, they will withdraw from the shared system. If they decide to share, they will try to game the management system by misreporting their resource demands to improve their performance, perhaps at the expense of others in the system. To address this challenge and study strategic behavior of self-interested users, game theory is known to be an effective tool. Drawing on game theory, this thesis encourages new thinking in designing management platforms robust to strategic behavior. In this thesis, we present five pieces of work on data center management platforms.

First, with the democratization of cloud and datacenter computing, users increasingly share large hardware platforms. In this setting, architects encounter two challenges: sharing fairly and sharing multiple resources. Drawing on game theory, we rethink fairness in computer architecture. A fair allocation must provide sharing incentives (SI), envy-freeness (EF), and Pareto efficiency (PE). We show that Cobb-Douglas utility functions are well suited to modeling user preferences for

cache capacity and memory bandwidth. Additionally we present an allocation mechanism that uses Cobb-Douglas preferences to determine each user's fair share of the hardware. This mechanism provably guarantees SI, EF, and PE, as well as strategy-proofness in the large (SPL). And it does so with modest performance penalties, less than 10% throughput loss, relative to an unfair mechanism.

Second, computational sprinting is a class of mechanisms that boost performance but dissipate additional power. We describe a sprinting architecture in which many independent chip multiprocessors share a power supply and sprints are constrained by the chips' thermal limits and the rack's power limits. Moreover, we present the computational sprinting game, a multi-agent perspective on managing sprints. Strategic agents decide whether to sprint based on application phases and system conditions. The game produces an equilibrium that improves task throughput for data analytics workloads by 4-6× over prior greedy heuristics and performs within 90% of an upper bound on throughput from a globally optimized policy.

Third, ensuring fairness in a system with scarce and commonly preferred resources requires time sharing. We consider a heterogeneous system with a few "big" and many "small" processors. We allocate heterogeneous processors using a novel token mechanism that supports game-theoretic notions of fairness such as sharing incentives and envy-freeness. The mechanism frames the allocation problem as a repeated game. In each round of the game, users request big processors and spend a token if their request is granted. We formulate game dynamics and optimize users' strategies to produce an equilibrium. Allocations from optimal strategies balance performance and fairness. Our token mechanism outperforms classical, fair mechanisms by 1.7x, on average, in total performance gains, and is competitive with a performance maximizing mechanism.

Fourth, we present a processor allocation framework that uses Amdahl's Law to model parallel performance and a market mechanism to allocate cores. We propose

the Amdahl utility function and demonstrate its accuracy when modeling performance from processor core allocations. We then design a market based on Amdahl utility and propose the Amdahl bidding procedure that optimizes users' bids for processors based on workload parallelizability. The framework uses entitlements to guarantee fairness yet outperforms existing proportional share algorithms.

Finally, sharing computational resources amortizes cost and improves utilization and efficiency. When agents pool their resources together, each becomes entitled to a portion of the shared pool. Static allocations in each round can guarantee entitlements and are strategy-proof, but efficiency suffers because allocations do not reflect variations in agents' demands for resources across rounds. Dynamic allocation mechanisms assign resources to agents across multiple rounds while guaranteeing agents their entitlements. Designing dynamic mechanisms is challenging, however, when agents are strategic and can benefit by misreporting their demands for resources.

The Amdahl bidding mechanism facilitates the trade in resources between users with static demands within a single management round. To facilitate the trade in resources between users with dynamic demands across multiple rounds, we propose two novel mechanisms. First, the T-period mechanism satisfies strategy-proofness and sharing incentives but with low efficiency. Second, the token mechanism satisfies strategy-proofness and guarantees at least a 50% approximation of sharing incentives, which means users receive at least half the utility they would have received by not participating in the mechanism. Through simulations on data gathered from Google clusters, we show that the performance of the token mechanism is comparable to that of state-of-the-art mechanisms that do not guarantee our game-theoretic properties. Further, although the token mechanism only guarantees a 50% approximation of sharing incentives, in practice, users receive at least 98% of their sharing incentives guarantee.

To the Promised One, the Last Rising Sun.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to start by thanking my adviser, Prof. Benjamin C. Lee, for all his help, support, and dedication during my PhD. I also want to extend my thanks to Prof. Vincent Conitzer, as this dissertation would not have been possible without his tremendous support, contributions, and mentorship. I would also like to thank all other members of my Prelim and PhD committees: Prof. Jeffrey S. Chase, Prof. Kamesh Munagala, Dr. Carl A. Waldspurger, Prof. Santiago Balseiro, and Prof. Peng Sun. I also want to express my gratitude to all my co-authors and collaborators, without whom I would not have been able to finish my PhD: Rupert Freeman, Songchun Fan, Qiuyun Llull, Derek R. Hower, Shivam Priyadarshi, Yuhao Li, Matthew Faw, Elijah Cole, Abhimanyu Yadav, Henri Maxime Demoulin, Zhiyu Zhang, and Paul Kim. And finally, I would like to thank my wife, my parents, my family, and my friends for their love and support.

1

Introduction

To improve efficiency and amortize cost over more computation, resource sharing has become vital in high performance computing systems [6]. In such systems, the conventional wisdom assumes that users have to share, regardless of the management policy. With a wide range of computing options available, this assumption does not seem to hold for today's self-interested users. These users selfishly pursue their individual performance without regard for others or the system. And if they dislike management outcomes, they will withdraw from the shared system. If they decide to share, they will try to game the management system by misreporting their resource demands to improve their performance, perhaps at the expense of others in the system.

Users' selfish behavior is not just a theoretical assumption. Previous work in systems literature has reported real-world examples of strategic behavior [7, 8, 9], making it a real challenge facing systems architects. To address this challenge and study strategic behavior of self-interested users, game theory is known to be an effective tool. Drawing on game theory, this thesis encourages new thinking in designing management platforms robust to strategic behavior. The main contributions are management platforms at different levels in datacenter systems: server processors [1, 10], server racks [2, 11], and server clusters [4, 5, 3].

1.1 Multi-resource Allocation in Server Processors [1]

In a shared server processor, computer architects encounter two challenges – sharing fairly and sharing multiple resources. To address these challenges, In Chapter 2, we propose Resource Elasticity Fairness (REF) [1, 10], a fair, multi-resource allocation mechanism that provably guarantees four fundamental game-theoretic properties. First, REF provides sharing incentives, ensuring that users perform no worse than under an equal division of resources. Second, REF provides envy-freeness, ensuring that each user prefers her own allocation over other users' allocations. Third, REF ensures Pareto efficiency, providing an allocation in which the system cannot improve a user's performance without harming another's. Finally, REF is strategy-proof when the number of users in a shared system is large, ensuring that users cannot improve their performance by misreporting their resource demands.

These properties are guaranteed when software preferences for hardware can be modeled by Cobb-Douglas utility functions. The Cobb-Douglas function accurately describes hardware performance for two fundamental reasons. First, it captures diminishing marginal returns in performance, a prevalent concept in computer systems. Second, the Cobb-Douglas function captures substitution effects, which are also typical – a user might trade off-chip memory bandwidth for last-level cache capacity. Using cycle-accurate simulations for diverse application suites, we show that Cobb-Douglas utility functions are well suited to modeling user utility for hardware resources.

1.2 Power Management in Server Racks [2]

In a datacenter rack, power supply is shared between servers. Most of today's servers are capable of computational sprinting by supplying extra power for short durations to enhance their performance. Although sprints improve servers' performance, uncoordinated sprints could overwhelm the rack's power supply and risk power emergencies. To maximize performance gains and minimize risks, systems architects face hard management questions – which servers should sprint and when should they sprint? In Chapter 3, we address these questions by designing the computational sprinting game [2]. In equilibrium, the game produces several desiderata – performance optimality of individual servers, system stability, and distributed sprinting management.

The sprinting architecture, which specifies the sprinting mechanism as well as power and cooling constraints, defines rules of the game. The game assumes that each server is controlled by a self-interested user who decides whether to sprint. Since simultaneous sprints could lead to power emergencies, users have to account for competitors' decisions before making any sprinting decision. When all users optimize their sprinting strategies against each other, the game reaches its equilibrium. To find an equilibrium, users make initial assumptions about system conditions and optimize their strategies. Doing so, they affect those same system conditions. Eventually, system conditions and users' strategies converge to a stationary distribution and the game reaches its equilibrium.

We show that users' equilibrium strategy is a simple threshold strategy – sprinting whenever utility gain exceeds a threshold. To find and maintain an equilibrium, we have proposed a computational sprinting management framework. Offline, the framework finds each user's sprinting threshold. Online, users decide whether to sprint by comparing a sprint's utility gain against their threshold. The framework permits distributed sprinting enforcement, because in equilibrium, users have no incentives to change their strategies.

1.3 Managing Heterogeneity in Server Clusters [3]

Ensuring fairness in a system with scarce and commonly preferred resources requires time sharing. To allocate processors in a datacenter with “big” and “small” processors, in Chapter 4, we devise a novel token mechanism that frames the allocation problem as a repeated game with discrete rounds [3]. At each round, users request big processors and spend a token if their request is granted. Spent tokens are then redistributed among users who do not receive a big processor. We formulate the game dynamics and optimized user strategies to produce an equilibrium. In equilibrium, allocations balance performance and fairness, outperforming fair mechanisms and being competitive with a performance maximizing mechanism. Allocations from optimal strategies balance performance and fairness. Our token mechanism outperforms classical, fair mechanisms by 1.7x, on average, in total performance gains, and is competitive with a performance maximizing mechanism.

1.4 Processor Core Allocation in Server Clusters [4]

In many private datacenters, users share a non-profit server cluster and its capital and operating costs. In such datacenters, a cluster manager must ensure users receive their entitlements, which specify the minimum share of resources each user should receive relative to others. For instance, in an academic cluster that combines servers purchased by researchers, entitlements may specify shares in proportion to researchers’ financial contributions.

Entitlements for processor cores in a datacenter differ from those in a server. Within a server, time on processor cores is a divisible resource that can be proportionally divided between users. The idealized datacenter provides a similar abstraction—

a warehouse-scale machine with a logically divisible pool of cores. However, cores are physically distributed across servers. This is challenging because users deploy different jobs on different servers, which means their demands for cores vary across servers. To address this challenge, a classical approach enforces proportional shares on each server separately, allocating each user her demand or entitlement, whichever is smaller. When entitlement exceeds demand, excess cores are redistributed among other users according to their entitlements. Although simple and widely used, this approach does not guarantee datacenter-wide entitlements.

To guarantee datacenter-wide entitlements, in Chapter 5, we design the Amdahl bidding mechanism [4]. The mechanism's centerpiece is the Amdahl utility function, which is derived from Amdahl's Law to model users' valuations for each server's cores. Users receive budgets in proportion to their entitlements and spend their budgets bidding for processor cores according to their Amdahl utility function. The market sets prices based on bids and users respond to prices until, in equilibrium, all cores are allocated and allocations are optimal. Informally, budgets satisfy entitlements while bids shift more resources to more parallelizable workloads. Market allocations are competitive with performance-centric ones. First, allocations incentivize sharing as each user always receives her entitlement and sometimes receives more. Second, allocations are Pareto-efficient, which means no other allocation can benefit one user without harming another. Third, the market is strategy-proof for highly competitive systems, which means no user can benefit by misreporting utility from processors. Finally, the market has low overheads as we have devised closed-form equations to calculate market allocations.

1.5 Allocate Resources across Time [5]

Sharing computational resources amortizes cost and improves utilization and efficiency. When agents pool their resources together, each becomes entitled to a portion of the shared pool. Static allocations in each round can guarantee entitlements and are strategy-proof, but efficiency suffers because allocations do not reflect variations in agents' demands for resources across rounds. Dynamic allocation mechanisms assign resources to agents across multiple rounds while guaranteeing agents their entitlements. Designing dynamic mechanisms is challenging, however, when agents are strategic and can benefit by misreporting their demands for resources.

In Chapter 6, we show that dynamic allocation mechanisms based on max-min fail to guarantee entitlements, strategy-proofness or both. We propose the flexible lending (FL) mechanism and show that it satisfies strategy-proofness and guarantees at least half the utility from static allocations while providing an asymptotic efficiency guarantee. Our simulations with real and synthetic data show that the performance of the flexible lending mechanism is comparable to that of state-of-the-art mechanisms, providing agents with at least 0.98x, and on average 15x, of their utility from static allocations. Finally, we propose the T-period mechanism and prove that it satisfies strategy-proofness and guarantees entitlements.

2

REF: Resource Elasticity Fairness with Sharing Incentives for Multiprocessors

2.1 Introduction

Datacenter platforms are often poorly utilized, running at less than 30% of peak capability [12]. With poor utilization, server power is amortized over little computation. To address this inefficiency, software must share hardware. Mechanisms for fair resource allocation (or a lack thereof) determine whether users have incentives to participate in dynamic, shared hardware platforms. In this setting, architects encounter two challenges: sharing fairly and sharing multiple resources.

We rethink fairness in resource allocation for computer architecture. Adopting the game-theoretic definition, a fair hardware allocation is one in which

- *all users perform no worse than under an equal division,*
- *no user envies the allocation of another, and*
- *no other allocation improves utility*
  without harming a user.

Our resource allocation strategy relies on robust game theory, encouraging users to share hardware and ensuring equitable allocations when they do. Conventional wisdom, on the other hand, assumes that users have no choice but to share. In this setting, prior efforts devise mechanisms to equally distribute performance penalties from sharing, which is not equitable [13, 14].

Drawing on economic game theory, we present a fair, multi-resource allocation mechanism. This mechanism and its resulting allocations provide key game-theoretic properties. First, the mechanism provides sharing incentives (SI), ensuring that each agent is at least as happy as they would be under an equal division of shared resources. Without SI, agents would not participate in the proposed sharing mechanism. Instead, they would rather equally and inefficiently divide the hardware. Supposing agents share a system, they will desire a fair division of the hardware.

In economic game theory, a fair allocation is defined to be envy-free (EF) and Pareto efficient (PE) [15]. An allocation is EF if each agent prefers his own allocation to other agents' allocations. Equitable sharing is defined by EF for all agents. An allocation is PE if we cannot improve an agent's utility without harming another agent.

Finally, a mechanism to allocate hardware should be strategy-proof (SP), ensuring that agents cannot gain by misreporting their preferences. Without SP, strategic agents may manipulate the hardware allocation mechanism by lying. In practice, SP may be incompatible with SI, EF, and PE [16]. But there exist allocation mechanisms that are approximately SP as long as many agents share a system. We refer to this weaker guarantee as strategy-proofness in the large (SPL).

Thus, we present a new framework for reasoning about fair resource allocation in computer architecture. Our contributions include the following:

- **Cobb-Douglas Utility in Computer Architecture.** We show that Cobb-

Douglas utility functions are well suited to model user performance and preferences for multiple hardware resources. Given Cobb-Douglas utilities, we detail conditions for SI, EF, and PE. (Section 2.3)

- **Fair Allocation for Computer Architecture.** We present a new mechanism to fairly allocate multiple hardware resources to agents with Cobb-Douglas utilities. We prove its game-theoretic properties (SI, EF, PE, SPL) and describe its implementation. (Section 2.4)
- **Case Study for Cache Size and Memory Bandwidth.** We apply the mechanism to fairly allocate cache size and memory bandwidth. We evaluate with cycle-accurate processor and memory simulators for diverse application suites, including PARSEC, SPLASH-2x, and Phoenix MapReduce. (Section 2.5)
- **Performance Trade-offs.** We compare our mechanism against prior approaches that equalize slowdown, describing how the latter violates game-theoretic fairness. Our mechanism provides fairness with modest penalties ($<$ 10% throughput loss) relative to a mechanism that does not provide SI, EF, PE, and SPL. (Section 2.5)

Without loss of generality, we evaluate our multi-resource allocation mechanism for cache size and memory bandwidth. In the future, the mechanism can support additional resources, such as the number of processor cores. Collectively, our findings establish robust foundations for the fair division of multiple hardware resources.

2.2 Motivation and Background

We present a mechanism for allocating shared resources. This mechanism guarantees SI, EF, PE, and SPL. And we demonstrate its ability to allocate last-level cache capacity and off-chip memory bandwidth. Our mechanism design relies on two fundamental insights about utility functions for computer architecture.

First, we use Cobb-Douglas utility functions to accurately capture hardware performance. For example, $u = x^{\alpha_x} y^{\alpha_y}$ models performance u as a function of resource allocations for cache capacity x and memory bandwidth y. The exponents α capture non-linear trends and model performance elasticity (i.e., sensitivity) for each hardware resource. For example, if $\alpha_x > \alpha_y$, the agent prefers cache capacity to memory bandwidth.

Second, we design a mechanism that uses each agent's reported resource elasticity α to determine his fair share of hardware. Given Cobb-Douglas utilities, the fair share can be expressed in a closed-form equation. Thus, the mechanism is computationally trivial. Yet the resulting allocation provably guarantees each of the desired game-theoretic properties: SI, EF, PE, and SPL.

Game-theoretic versus Heuristic Fairness. Our approach addresses fundamental limitations in prior work. Prior mechanisms consider each user's performance penalty incurred from sharing [17, 18]. They then allocate a resource, such as memory bandwidth, trying to equalize slowdown [14]. While this approach produces equal outcomes, it is not fair in the economic sense. Our rigorous, game-theoretic analysis shows that equalizing slowdown provides neither SI nor EF.

Without these properties, strategic users would have no incentive to share. They would prefer an equal division of memory bandwidth rather than receive an equal slowdown guarantee from the allocation mechanism. Allocating multiple resources with heuristics, such as hill-climbing [19], is even more difficult and provides even fewer guarantees.

Cobb-Douglas versus Leontief. Cobb-Douglas allows us to guarantee fairness in computer architecture for the first time. Although Leontief [8, 20, 21] provides the same guarantees in distributed systems, they do not apply in a more fine-grained analysis of hardware behavior for two reasons.

First, unlike Leontief, Cobb-Douglas utilities capture diminishing returns and substitutability. Both of these effects are prevalent in architecture, whether in Amdahl's Law for multi-core parallelism [22], in data locality for cache sizing, or in communication intensity for bandwidth allocation. In these settings, linear Leontief preferences of the form $u = min(x_1/\alpha_1, x_2/\alpha_2)$ are ineffective.

Second, consider the complexity of Cobb-Douglas and Leontief. We use classical regression to fit log-linear Cobb-Douglas to architectural performance. In contrast, since Leontief is concave piecewise-linear, fitting it would require non-convex optimization, which is computationally expensive and possibly NP-hard [23]. Note that [8, 20, 21] did not encounter these difficulties because they assume that agents in a distributed system provide a demand vector (e.g., 2CPUs, 4GB-DRAM). Fitting architectural performance to Leontief is equivalent to finding the demand vector for substitutable microarchitectural resources (e.g., cache and memory bandwidth), which is conceptually challenging.

2.3 Fair Sharing and Cobb-Douglas

A mechanism for fair sharing should guarantee several game theoretic properties. First, the mechanism must provide sharing incentives (SI). Without such incentives, software agents would prefer equally divided resources to a sophisticated mechanism that shares hardware more efficiently.

If agents do intelligently share, they will want a fair division. In economic game theory, a fair allocation is envy-free (EF) and Pareto efficient (PE) [15]. We present an allocation mechanism that provides SI, EF, and PE for hardware resources given software agents with Cobb-Douglas utility.

Cobb-Douglas Utility. Suppose multiple agents share a system with several types of hardware resources $1, \ldots, R$. Let $x_i = (x_{i1}, \ldots, x_{iR})$ denote agent i's hardware allocation. Further, let $u_i(x_i)$ denote agent i's utility. Equation (2.1) defines utility within the Cobb-Douglas preference domain.

$$u_i(x_i) = \alpha_{i0} \prod_{r=1}^{R} x_{ir}^{\alpha_{ir}} \tag{2.1}$$

The exponents α introduce non-linearity, useful for capturing diminishing marginal returns in utility. The product models interactions and substitution effects between resources. The user requires both resources for progress because utility is zero when either resource is unavailable.

The parameters $\alpha_i = (\alpha_{i1}, \ldots, \alpha_{iR})$ quantify the elasticity with which an agent demands a resource. If $\alpha_{ir} > \alpha_{ir'}$, then agent i benefits more from resource r than from resource r'. These parameters are tailored to each agent and define her demand for resources.

With Cobb-Douglas utility functions, we reason about agents' preferences. Consider two allocations x and x' for agent i.

- If $u_i(x) > u_i(x')$, then $x \succ_i x'$ (strictly prefer x to x')
- If $u_i(x) = u_i(x')$, then $x \sim_i x'$ (indifferent to x and x')
- If $u_i(x) \geq u_i(x')$, then $x \succsim_i x'$ (weakly prefer x to x')

Cobb-Douglas preferences are a good fit for resources in computer architecture. They capture diminishing marginal returns and substitution effects in ways that linear Leontief preferences, which prior work uses [8], cannot.

Example with Cache and Memory. We use a recurring example to illustrate the allocation of multiple resources given Cobb-Douglas preferences. Consider processor cache size and memory bandwidth. Agents see diminishing marginal returns from larger caches since software tasks exhibit limited exploitable locality. Depending on its data access locality, software tasks can substitute cache size for memory bandwidth and vice versa.

Suppose a system has 24GB/s of memory bandwidth and 12MB cache. This setting is representative of a quad-core processor with two DDRX channels. The system is shared by two users or agents. Let (x_1, y_1) denote the memory bandwidth and cache capacity allocated to the first user. Similarly, let (x_2, y_2) denote the second user's allocation. Suppose users' utilities are described by Equation (2.2).

$$u_1 = x_1^{0.6} y_1^{0.4} \qquad u_2 = x_2^{0.2} y_2^{0.8} \tag{2.2}$$

User 1 runs an application that exhibits bursty memory activity but little data re-use. For user 1, memory bandwidth x_1 is more useful than cache capacity y_1. In contrast, user 2 makes good use of its cache capacity y_2. We use profilers and regression to derive these utility functions (Section 2.4.4).

Software behavior translates into hardware demands, which in turn are reflected in the utility functions. These utility functions are representative of realistic applications. For example, u_1 and u_2 accurately model the relative cache and memory intensities for `canneal` and `freqmine` from the PARSEC benchmarks (Section 2.5).

Visualization with Edgeworth Boxes. To visualize feasible resource allocations, we use the Edgeworth box [24]. Figure 2.1 illustrates the allocation of two resources to two users. User 1's origin is at the lower left corner and User 2's origin is at the upper right corner. The total amount of cache is the height of the box and the total amount of memory bandwidth is the width. Therefore, each feasible allocation of resources can be represented as a point in the Edgeworth box. If user 1 gets 6GB/s memory bandwidth and 8MB cache, user 2 is left with 18GB/s memory bandwidth and 4MB cache.

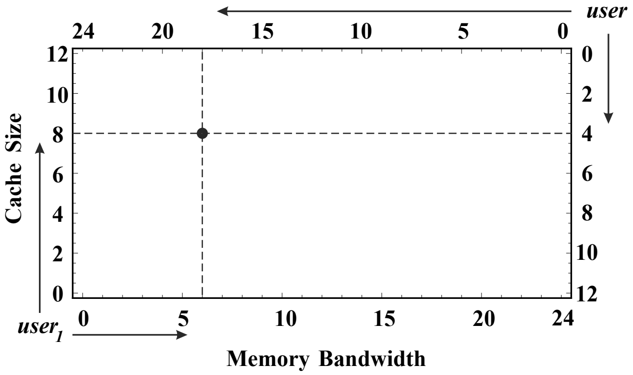


FIGURE 2.1: **Edgeworth Box Example.** Box height shows total cache size and box width shows total memory bandwidth. Each point in this box corresponds to a feasible resource allocation to users.

The Edgeworth box includes all possible allocations. But only some of these allocations are fair. And only some of these provide sharing incentives. Thus, desired game-theoretic properties (sharing incentives, envy-freeness, and Pareto efficiency) define constraints on the allocation space. We use the Edgeworth box to visualize these constraints, beginning with sharing incentives.

2.3.1 Sharing Incentives (SI)

Sharing hardware is essential to increasing system utilization and throughput. An allocation mechanism should provide sharing incentives (SI) such that agents are at least as happy as they would be under an equal split of the resources. Without SI, users would prefer to partition hardware equally. But an equal partitioning would not reflect software diversity and heterogeneous hardware demands. Resources may be mis-allocated, leaving throughput unexploited.

Formally, let C_r denote the total capacity of resource r in the system. Suppose an allocation mechanism provides agent i with resources $x_i = (x_{i1}, \ldots, x_{iR})$. For a system with N users, this mechanism provides SI if

$$(x_{i1}, \ldots, x_{iR}) \succsim_i \left(\frac{C_1}{N}, \ldots, \frac{C_R}{N} \right) \tag{2.3}$$

for each agent $i \in [1, N]$. In other words, each agent weakly prefers its allocation of hardware to an equal partition.

Whether an allocation is preferred depends on the utility functions. Consider our example with cache size and memory bandwidth. User 1 compares its allocation (x_1, y_1) against equally splitting 24GB/s of bandwidth and 12MB of cache. If user 1 always weakly prefers (x_1, y_1), then the allocation mechanism provides user 1 an incentive to share.

$$x_1^{0.6} y_1^{0.4} \geq \left(\frac{24\text{GB/s}}{2} \right)^{0.6} \left(\frac{12\text{MB}}{2} \right)^{0.4} \tag{2.4}$$

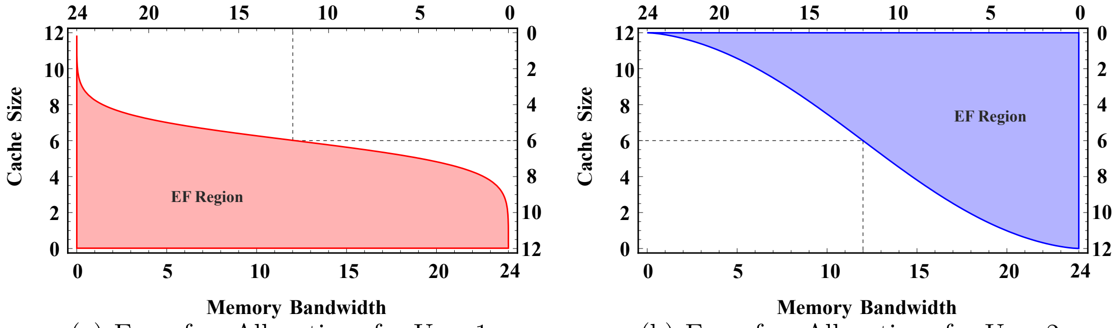
$$x_2^{0.2} y_2^{0.8} \geq \left(\frac{24\text{GB/s}}{2} \right)^{0.2} \left(\frac{12\text{MB}}{2} \right)^{0.8} \tag{2.5}$$

In our example with two agents, Equations (2.4)–(2.5) must be satisfied. User 1 must receive allocations that satisfy Equation (2.4). Simultaneously, user 2 must receive allocations that satisfy Equation (2.5). A mechanism that provides SI will identify allocations that satisfy both constraints.

2.3.2 Envy-Freeness (EF)

Envy is the resentment of another agent's allocation combined with a desire to receive that same allocation. Allocations are envy-free (EF) if no agent envies another. Such allocations are considered equitable and equity is a game-theoretic requirement for fairness [15].

Specifically, suppose agent i is allocated x_i. This allocation is EF if agent i prefers its allocation to any other agent's allocation and has no desire to swap. That



(a) Envy-free Allocations for User 1

(b) Envy-free Allocations for User 2

Figure 2.2: **Visualizing Envy-freeness (EF).** The mid-point and two corners, and are always EF.

is, $x_i \succsim_i x_j, \forall j \neq i$. In this comparison, each agent considers herself in the place of other agents and evaluates their allocations in the same way she judges her own allocation.

In our cache and bandwidth example, the EF allocations for user 1 are those for which $u_1(x_1, y_1) \geq u_1(x_2, y_2)$. Note that $(x_2, y_2) = (24-x_1, 12-y_1)$. Thus, allocations that satisfy Equation (2.6) are EF for user 1. And Figure 2.2(a) illustrates regions in which these allocations are found. Similarly, Equation (2.7) and Figure 2.2(b) describe the set of EF allocations for user 2. A mechanism that satisfies EF will identify allocations that satisfy both constraints.

$$x_1^{0.6} y_1^{0.4} \geq (24 - x_1)^{0.6} (12 - y_1)^{0.4} \tag{2.6}$$

$$x_2^{0.2} y_2^{0.8} \geq (24 - x_2)^{0.2} (12 - y_2)^{0.8} \tag{2.7}$$

There are always at least three EF allocations, which are illustrated by the middle point and two corner points. The middle point corresponds to the situation in which all resources all equally divided between users. No user envies the other.

The corners correspond to situations in which all of one resource is given to one user and all of the other resource is given to the other. Both users derive zero utility and do not envy each other. In our example, the two corner allocations are

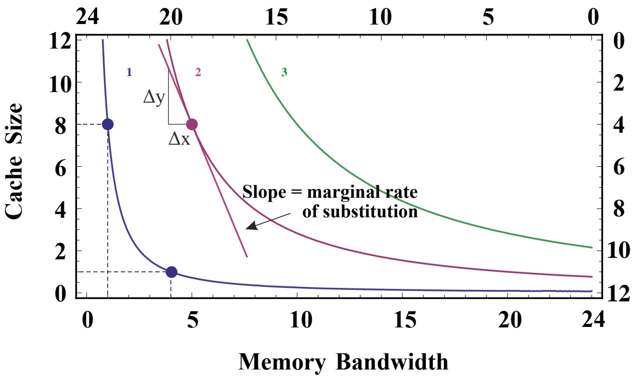


FIGURE 2.3: **Cobb-Douglas Indifference Curves.** On a given indifference curve, allocations provide the same utility.

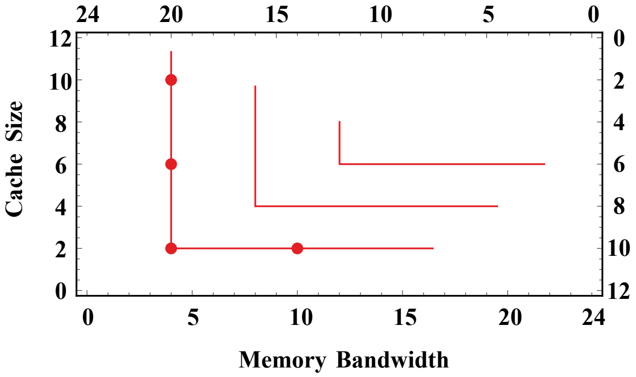


FIGURE 2.4: **Leontief Indifference Curves.** Resources are perfect complements and the marginal rate of substitution is either zero or infinity.

(0GB/s, 12MB) and (24GB/s, 0MB). Users derive zero utility because both cache and memory are required for computation.

None of these obvious EF allocations is attractive. The middle point divides resources equally without accounting for differences in user utility. In this setting, system throughput could likely be improved. And corner points are clearly not useful. Thus, we need a mechanism to identify more effective EF allocations.

2.3.3 Pareto Efficiency (PE)

Pareto efficiency (PE) is another game-theoretic property that must be satisfied by a fair resource allocation [15]. An allocation is PE if increasing one user's utility necessarily decreases another's utility. If an allocation is not PE, there exists another allocation that should have been chosen to improve total system utility.

More precisely, consider an allocation $x = (x_1, \ldots, x_N)$ for N agents. Allocation x is PE if there exists no other feasible allocation x' that all agents i weakly prefer ($x'_i \succsim_i x_i$) and at least one agent j strictly prefers ($x'_j \succ_j x_j$). Finding PE allocations is inherently linked to navigating trade-offs between substitutable resources.

Substitution Effects. An indifference curve depicts the allocations that are substitutable for one another. Figure 2.3 shows three indifference curves for user 1. Allocations on the same curve provide the same utility. Allocations on different curves provide different utilities. The utility of I_1 is less than that of I_2, and the utility of I_2 is less than that of I_3. Therefore, all allocations on I_2 and I_3 are strictly preferred to those on I_1.

The Leontief preferences used in prior work do not permit substitution [8]. Suppose user 1 demands 2GB/s of memory bandwidth and 1MB of cache. With this demand vector, the user's Leontief utility function is shown in Equation (2.8). Under Leontief, resources are perfect complements, leading to the L-shaped indifference curves in Figure 2.4.

$$u_1 = min\{x_1, 2y_1\} \tag{2.8}$$

User 1 demands bandwidth and cache in a 2:1 ratio. If the allocated ratio differs, then extra allocated resources are wasted. For example, user 1 derives the same utility from (4GB/s, 2MB) as it does from disproportional allocations such as (10GB/s, 2MB) or (4GB/s, 10MB). Leontief preferences do not account for marginal benefits from disproportional allocations. Nor do they allow for substitution in which more cache capacity compensates for less memory bandwidth.

In contrast, substitution is modeled by Cobb-Douglas preferences as illustrated

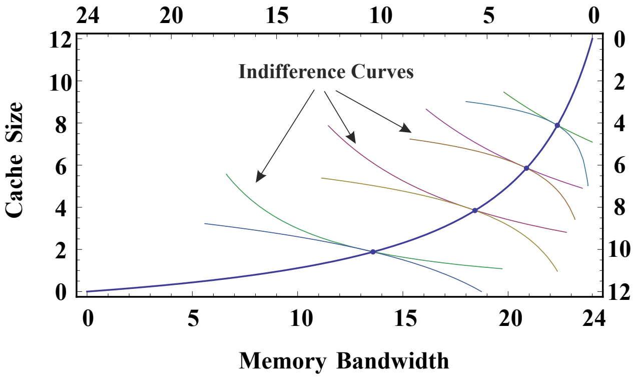


FIGURE 2.5: **Visualizing Pareto Efficiency.** The contract curve includes all PE allocations for which MRS for both utility functions are equal.

by indifference curves' slopes in Figure 2.3. For instance, user 1 can substitute an allocation of (4GB/s, 1MB) for an allocation of (1GB/s, 8MB). Such flexibility provides the allocation mechanism with more ways to provide the same utility, which is particularly important as the set of feasible allocations are constrained by the conditions for SI, EF, and PE.

Marginal Rates of Substitution. The marginal rate of substitution (MRS), is the rate at which the user is willing to substitute one resource for the other. Visually, the MRS is the slope of the indifference curve. If MRS=2, the user will give up two units of y for one unit of x. Under Leontief preferences, the MRS is either zero or infinity; the user has no incentive for substitution. But under Cobb-Douglas preferences, the MRS is more interesting. In our cache and bandwidth example, the marginal rate of substitution for user 1 is given by Equation (2.9).

$$MRS_{1,xy} = \frac{\partial u_1/\partial x_1}{\partial u_1/\partial y_1} = \left(\frac{0.6}{0.4}\right)\left(\frac{y_1}{x_1}\right) \tag{2.9}$$

For any PE allocation, the MRS for the two users must be equal. Visually, this means users' indifference curves are tangent for PE allocations. Suppose curves were not tangent for a particular allocation. Then a user i could adjusts its allocation and travel along its indifference curve, substituting resources based on its MRS without affecting u_i. But the substitution would take the other user to a higher utility.

The MRS determines the contract curve, which shows all PE allocations. Figure 2.5 shows the contract curve and illustrates tangency for three allocations. With the tangency condition, formal conditions for PE is easily formulated. In our example, allocations (x_1, y_1) and (x_2, y_2) are PE if the users' marginal rates of substitution are equal; Equation (2.10) must be satisfied.

$$\left(\frac{0.6}{0.4}\right)\left(\frac{y_1}{x_1}\right) = \left(\frac{0.2}{0.8}\right)\left(\frac{y_2}{x_2}\right) \tag{2.10}$$

As seen in Figure 2.5, both origins are PE allocations. At these points, one user's utility is zero and the other's is maximized. Increasing a user's utility, starting from zero, necessarily decreases the other user's utility. While PE, these allocations are neither desirable nor fair. The user with zero utility envies the other user's allocation. Thus, we need a mechanism that identifies both PE and EF allocations.

2.4 Resource Elasticity Fairness (REF)

We present a fair allocation mechanism that satisfies three game-theoretic properties: sharing incentives (SI), envy-freeness (EF), and Pareto efficiency (PE). We begin with the space of possible allocations. We then add constraints to identify allocations with the desired properties.

Economic game theory defines a fair allocation as one that is equitable (EF) and efficient (PE) [15]. Figure 2.6 illustrates the effect of these constraints. Each user identifies its EF allocations. And the contract curve identifies PE allocations. The intersection of these three constraints define feasible, fair allocations. Figure 2.7 shows that SI further constrains the set of fair allocations.

Formally, finding fair multi-resource allocations given Cobb-Douglas preferences

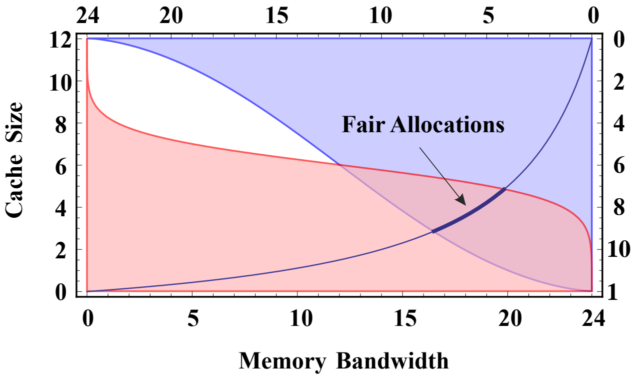


FIGURE 2.6: **Fair Allocation Set.** All the points on the intersection of envy-free sets and the contract curve correspond to the fair allocations.

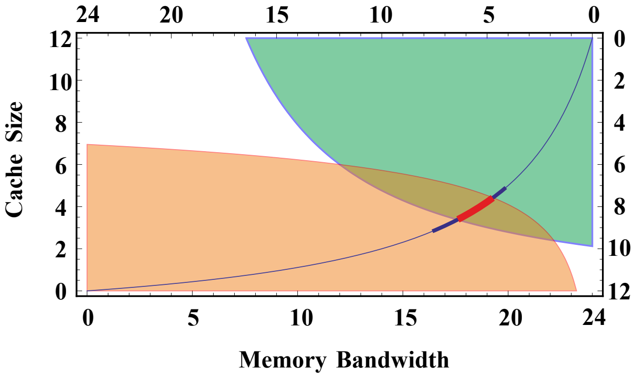


FIGURE 2.7: **Visualizing Sharing Incentives.** Satisfying the sharing incentive property limits the set of feasible fair allocations.

can be modeled as the following feasibility problem for N agents and R resources.

$$\begin{aligned}
&\text{find} && x && && (2.11)\\
&\text{subject to} && u_i(x_i) \geq u_i(x_j) && i,j{\in}[1,N]\\
& && \frac{\alpha_{ir}}{\alpha_{is}}\frac{x_{is}}{x_{ir}} = \frac{\alpha_{jr}}{\alpha_{js}}\frac{x_{js}}{x_{jr}} && i,j{\in}[1,N];r,s{\in}[1,R]\\
& && u_i(x_i) \geq u_i(C/N) && i{\in}[1,N]\\
& && \sum_{i=1}^{N} x_{ir} \leq C_r, && r{\in}[1,R]
\end{aligned}$$

where C/N is $(C_1/N, \ldots, C_R/N)$. In this formulation, the four constraints enforce

EF, PE, SI, and capacity.

2.4.1 Procedure for Fair Allocation

To solve the multi-resource allocation problem, we present a mechanism to determine each agent's fair share of the hardware. N agents share R resources. For each agent i, we determine its allocation $x_i = (x_{i1}, \ldots, x_{iR})$ with the following procedure, which satisfies all constraints in Equation (2.11).

- **Fit Cobb-Douglas Utility.** Profile and characterize agent i's performance for various resource allocations. Fit a Cobb-Douglas utility function $u_i(x_i) = \alpha_{i0} \prod_{r=1}^{R} x_{ir}^{\alpha_{ir}}$.

- **Re-scale Elasticities.** Parameters α in the Cobb-Douglas utility function are known as elasticities. For each agent i, re-scale its elasticities so that they sum to one.

$$\hat{\alpha}_{ir} = \frac{\alpha_{ir}}{\sum_{r=1}^{R} \alpha_{ir}} \tag{2.12}$$

- **Re-scale Utilities.** Redefine the Cobb-Douglas utility function with re-scaled elasticities $\hat{u}_i(x_i) = \prod_{r=1}^{R} x_{ir}^{\hat{\alpha}_{ir}}$.

- **Allocate in Proportion to Elasticity.** Examine re-scaled Cobb-Douglas utilities and use their elasticities to determine fair share for each agent i and resource r.

$$x_{ir} = \frac{\hat{\alpha}_{ir}}{\sum_{j=1}^{N} \hat{\alpha}_{jr}} \times C_r \tag{2.13}$$

In effect, this allocation mechanism quantifies elasticity α to determine the extent each resource improves an agent's utility. Re-scaling elasticities allows us to compare values for different agents on the same scale. By allocating in proportion to elasticity, agents that benefit more from resource r will receive a larger share of the total C_r.

In our cache and bandwidth example, two users provide Cobb-Douglas utility functions with elasticities. These elasticities are already scaled and sum to one (e.g., $u_1 = x_1^{0.6} y_1^{0.4}$). To determine the memory bandwidth allocation, we examine both user's bandwidth elasticity ($\alpha_{1x} = 0.6, \alpha_{2x} = 0.2$) and allocate proportionally.

$$x_1 = \left(\frac{0.6}{0.8}\right) \times 24 = 18\text{GB/s}, \qquad y_1 = \left(\frac{0.4}{1.2}\right) \times 12 = 4\text{MB}$$

$$x_2 = \left(\frac{0.2}{0.8}\right) \times 24 = 6\text{GB/s}, \qquad y_2 = \left(\frac{0.8}{1.2}\right) \times 12 = 8\text{MB}$$

2.4.2 Fairness and Sharing Incentives

The proportional elasticity mechanism has several attractive properties. The mechanism promotes sharing and guarantees fairness by satisfying conditions for SI, EF, and PE. We sketch the proofs for these properties.

First, we show that the allocation is a Nash bargaining solution. Observe that the allocation from Equation (2.13) is equivalent to finding an allocation that maximizes the product of re-scaled utilities $\hat{u}$. This equivalence can be shown by substituting re-scaled Cobb-Douglas utility functions into Equation (2.14) and using Lagrange multipliers for constrained optimization.

$$\max \prod_{i=1}^{N} \hat{u}_i(x_i) \quad \text{subject to} \quad \sum_{i=1}^{N} x_{ir} \leq C_r \tag{2.14}$$

In game theory, the bargaining problem asks how agents should cooperate to produce Pareto efficient outcomes. Nash's solution is to maximize the product of utilities [25, 26], which is equivalent to Equation (2.14) and our allocation mechanism. Thus, our mechanism produces an allocation that is also a Nash bargaining solution.

Next, we show that our allocation is also a Competitive Equilibrium from Equal Outcomes (CEEI), a well-known microeconomic concept for fair division. In CEEI, users are initially assigned equal resource allocations. Based on user preferences, prices are assigned to resources such that users trade and the market clears to produce an allocation.

The CEEI solution picks precisely the same allocation of resources as the Nash bargaining solution for homogeneous utility functions [27]. Let $x = (x_1, \ldots, x_R)$ be a vector of resources. Utility function u is homogeneous if $u(kx) = ku(x)$ for some constant k. Our re-scaled Cobb-Douglas utilities are homogeneous because $\sum_{r=1}^{R} \hat{\alpha}_r = 1$. For this reason, our allocation is a solution to both the Nash bargaining problem and CEEI.

Finally, a CEEI allocation is known to be fair, satisfying both EF and PE [15]. CEEI solutions also satisfy SI because users start with an equal division of resources. Users would only deviate from this initial division if buying and selling resources in the CEEI market would increase utility. Thus, users can do no worse than an equal division and CEEI provides SI.

In summary, our allocation mechanism is equivalent to the Nash bargaining solution, which is equivalent to the CEEI solution. Because the CEEI solution provides SI, EF, and PE for re-scaled Cobb-Douglas utility functions, the proportional elasticity mechanism provides these properties as well.

2.4.3 Fairness and Strategy-Proofness in the Large

The proportional elasticity mechanism is strategy-proof in the large. An allocation mechanism is strategy-proof (SP) if a user cannot gain by mis-reporting its utility functions. Unfortunately, SP is too restrictive a property for Cobb-Douglas utility functions. For these preferences, no mechanism can provide both PE and SP [16]. However, our mechanism does satisfy a weaker property, strategy-proofness in the large (SPL). When there are many users in the system, users have no incentive to lie about their elasticities α.

First, we define large. A large system has many users such that the sum of all agents' elasticities for any resource is much bigger than 1. In such a system, any one user's resource elasticity is small relative to the sum of all agents' elasticities for the resource. More formally, the system is large if $1 \ll \sum_j \alpha_{jr}$, for all resources r.

Next, suppose user i decides to lie about her utility function, reporting α'_{ir} instead of the true value α_{ir} for resource r. Given other users' utilities, user i would choose to report the α'_{ir} that maximizes her utility.

$$\frac{\partial}{\partial \alpha'_{ik}} \prod_{r=1}^{R} \left(\frac{\alpha'_{ir}}{\alpha'_{ir} + \sum_{j \neq i} \alpha_{jr}} C_r \right)^{\alpha_{ir}} = 0 \quad \forall k \in [1, R] \tag{2.15}$$

In her best scenario, user i knows all other users' utilities and $\alpha_{jr}, \forall j \neq i$. Thus, by mis-reporting α'_{ir}, user i can precisely affect her proportional share of resource r. Yet, when user i receives her allocation, she evaluates it with α_{ir}, which reflects her true utility from resource r. Thus, the product in Equation (2.15) reflects user i's utility from lying.

User i attempts to maximize this utility from lying, taking partial derivatives with respect to α'_{ir}. But it can be proven that this optimization produces $\alpha'_{ir} \approx \alpha_{ir}$ when $1 \ll \sum_j \alpha_{jr}$ for all resources r.[1] Thus, in a large system, our allocation mechanism is approximately strategy proof. A user cannot benefit by lying about her utility.

In theory, SPL holds when an individual agents elasticity is much smaller than the sum of all agents elasticities. In practice, we find that tens of agents are sufficient to provide SPL. In other words, a strategic agent performing the optimization of Equation (2.15) will not deviate from her true elasticity.

[1] See Appendix A.1 for the proof

For example, consider 64 tasks sharing a large system. This is a realistic setting since modern servers can have four processor sockets (= 64 threads) that share eight-twelve memory channels (> 100 GB/s of bandwidth). Suppose each of the 64 tasks elasticities are uniformly random from (0,1). We analyze Equation (2.15) and find that SPL holds.

2.4.4 Implementing the Mechanism

To implement the proportional elasticity mechanism, we need Cobb-Douglas utilities. We describe the process for deriving these utilities based on performance profiles and statistical regression. We also describe how proportional shares can be enforced by leveraging known resource schedulers.

Profiling Performance. Suppose a user derives utility from performance. Without loss of generality, we measure performance as the number of instructions committed per cycle (IPC). Execution time, speed-ups over a baseline, and energy efficiency would all exhibit similar trends.

The user profiles its performance as a function of allocated resources. These profiles reveal the rate of diminishing returns and identify resource substitutability. For example, the user samples from the allocation space to determine sensitivity to cache size and memory bandwidth. These profiles provide the data needed to derive utilities.

Performance can be profiled in several ways. First, consider off-line profiling in which a user runs software while precisely varying the available hardware. For example, a user can co-locate its task with synthetic benchmarks that exert tunable pressure on the memory hierarchy [28]. Thus, profiles would quantify cache and bandwidth sensitivity.

Also off-line, the user might rely on cycle-accurate, full-system simulators. These simulators combine virtual machines, such as QEMU, with hardware timing models to accurately model processor and memory [29, 30]. Simulated and physical hardware may report different performance numbers. But simulators can accurately report trends and elasticities, identifying hardware resources that are more important for performance. We value relative accuracy over absolute accuracy when profiling hardware preferences.

Finally, consider on-line profiling. Without prior knowledge, a user assumes all resources contribute equally to performance. Such a naive user reports utility $u = x^{0.5}y^{0.5}$. As the system allocates for this utility, the user profiles software performance. And as profiles are accumulated for varied allocations, the user adapts its utility function.

Fitting Cobb-Douglas Utility. Given performance profiles for varied hardware allocations, each user fits her Cobb-Douglas utility function in the form of $u = \alpha_0 \prod_{r=1}^{R} x_r^{\alpha_r}$. For example, let u be IPC, let x_1 be cache capacity, and let x_2 be memory bandwidth.

Fitting the utility function means identifying elasticities $\alpha = (\alpha_0, \ldots, \alpha_R)$ that best relate performance to the resources. We fit α with regression. Specifically, we apply a log transformation to linearize Cobb-Douglas. After this transformation, we have a standard linear model with parameters α as shown in Equation (2.16). Parameters are fit with least squares.

$$\log(u) \quad = \quad \log(\alpha_0) + \sum_{r=1}^{R} \alpha_r \log(x_r) \tag{2.16}$$

Allocating Proportional Shares. We re-scale elasticities from each user's Cobb-Douglas utility function and compute proportional shares. The novelty of our mechanism is not in proportional sharing but in how we identify the proportions based on Cobb-Douglas elasticities to ensure SI, EF, and PE. After the procedure determines proportional shares for each user, we can enforce those shares with existing approaches, such as weighted fair queuing [31] or lottery scheduling [32].

2.4.5 Alternative Fair Mechanisms

There may exist multiple allocations x that satisfy the fairness conditions in Equation (2.11). Our mechanism for proportional elasticity is only one possible mechanism for one possible solution. Alternative mechanisms may also produce fair allocations but increase computational complexity. Suppose we follow prior work in computer architecture and seek fair allocations that maximize system throughput.

To evaluate throughput for a multi-programmed system, architects define the notion of weighted progress, which divides each application's multi-programmed IPC by its single-threaded IPC [17]. Weighted system throughput is the sum of each user's weighted progress. This is the metric used to evaluate prior work on memory scheduling and multiprocessor resource management [33, 19].

$$\sum_{i=1}^{N} \frac{\mathrm{IPC}(x_i)}{\mathrm{IPC}(C)} \quad \approx \quad \sum_{i=1}^{N} \frac{u_i(x_i)}{u_i(C)} \quad = \quad \sum_{i=1}^{N} U(x_i) \tag{2.17}$$

We adapt this notion of normalized throughput, expressing it in terms of our utility functions. This means dividing utility for an allocation in the shared machine $u_i(x_i)$ by utility when given all of the machine's capacity $u_i(C)$. Let $U(x_i) = u_i(x_i)/u_i(C)$ define the notion of weighted utility, which is equivalent to the notion of slowdown in prior work [33, 19].

Fair Allocation for Utilitarian Welfare. Rather than allocate in proportion to elasticities, we could allocate to maximize utilitarian welfare. Instead of finding x subject to fairness conditions in Equation (2.11), we would optimize $\max \sum_i U_i(x_i)$ subject to the same conditions. While $\max \sum_i U_i(x_i)$ is computationally intractable, max $\prod_i U_i(x_i)$ is similar but tractable with geometric programming.[2] But this mechanism would be more computationally demanding than our closed-form solution in Equation (2.13).

Yet a utilitarian mechanism is interesting. Overall system performance is an explicit optimization objective. A utilitarian mechanism likely provides the allocation that achieves the highest performance among all fair allocations. In effect, utilitarian allocations provides an empirical upper bound on fair performance.

Fair Allocation for Egalitarian Welfare. We could also find fair allocations to optimize egalitarian welfare. In Equation (2.11), we would optimize max-min $U_i(x_i)$ subject to fairness conditions. As before, geometric programming can perform this optimization but this mechanism would be more computationally demanding than our closed-form solution.

Egalitarian welfare is interesting because it optimizes for the least satisfied user. EF and PE define conditions for a fair allocation. But these conditions say nothing about equality in outcomes. An allocation could be fair but the difference between the most and least satisfied user in the system could be large. The max-min optimization objective mitigates inequality in outcomes, perhaps at the expense of system welfare. Egalitarian allocations might provide an empirical lower bound on fair performance.

Unfair Allocation. Finally, we could neglect game-theoretic fairness and ignore constraints imposed by SI, EF, and PE. In this setting, we would maximize welfare subject only to capacity constraints. Note that optimizing egalitarian welfare without fairness conditions is equivalent to the objective in prior work [33], which equalizes users' weighted progress such that $\max_i\ U_i(x_i)\ /\ \min_j\ U_j(x_j) \rightarrow 1$. The max-min objective for egalitarian welfare causes the denominator to approach the numerator. Assessing performance of unfair allocations reveals the penalty we must pay for SI,

[2] Cobb-Douglas is a monomial function (i.e., function with the form $f(x) = ax_1^{\alpha_1}x_2^{\alpha_2}, \ldots, x_m^{\alpha_m}$). And geometric programming can maximize monomials [34].

Table 2.1: Platform Parameters

Component	**Specification**
Processor	3 GHz OOO cores, 4-width issue and commit
L1 Cache	32 KB, 4-way set associative, 64-byte block size, 2-cycle latency
L2 Cache	[128 KB, 256 KB, 512 KB, 1 MB, 2 MB], 8-way set associative, 64-byte block size, 20-cycle latency
DRAM Controller	Closed-page, Queue per rank, Rank then bank round-robin scheduling
DRAM Bandwidth	[0.8 GB/s, 1.6 GB/s, 3.2 GB/s, 6.4 GB/s, 12.8 GB], single channel

EF, and PE.

2.5 Evaluation

We evaluate the proportional elasticity mechanism when sharing the last-level cache and main memory bandwidth in a chip-multiprocessor. In this setting, we evaluate several aspects of the mechanism. First, we show that Cobb-Douglas utilities are a good fit for performance. Then, we interpret utility functions to identify applications that prefer cache capacity (C) and memory bandwidth (M).

Finally, we compare the proportional elasticity mechanism against an equal slowdown mechanism, which represents conventional wisdom. We find that equal slowdown fails to guarantee game-theoretic fairness. On the other hand, proportional elasticity guarantees SI, EF and PE with only modest performance penalties relative to an unfair approach.

2.5.1 Experimental Methodology

Simulator. We simulate the out-of-order cores using the MARSSx86 full system simulator [29]. We integrate the processor model with the DRAMSim2 simulator [30] to simulate main memory. To characterize application sensitivity to allocated cache size and memory bandwidth, we simulate 25 architectures spanning combinations of five cache sizes and five memory bandwidths. The platform parameters are described in Table 2.1.

Given simulator data, we use Matlab to fit Cobb-Douglas utility functions. Our mechanism includes a closed-form expression for each agent's fair allocation. But to evaluate other mechanisms that require geometric programming, we use CVX [35], a convex optimization solver.

Workloads. We evaluate our method on 24 benchmarks from PARSEC and SPLASH-2x suites [36]. We further evaluate applications from the Phoenix system for MapReduce programming [37], including histogram, linear regression, string match, and word count. For PARSEC 3.0 benchmarks, we simulate 100M instructions from the regions of interest (ROI), which are representative application phases identified by MARSSx86 developers. Phoenix applications we simulate 100M instructions from the beginning of the map phase.

2.5.2 Fitting Cobb-Douglas Utility

Each application is associated with a user. Application performance is measured as instructions per cycle (IPC). Using cycle-accurate simulations, we profile each benchmark's performance. Given these profiles for varied cache size and memory bandwidth allocations, we perform a linear regression to estimate utility functions.

For each application, we use a Cobb-Douglas utility function $u = \alpha_0 x^{\alpha_x} y^{\alpha_y}$ where u is application performance measured with IPC, x is memory bandwidth, and y is cache size. Although a non-linear relationship exists between Cobb-Douglas utility and resource allocations, a logarithmic transformation produces a linear model (Equation (2.16)). Least squares regression estimates the resource elasticities α for each benchmark.

To evaluate this fit, we report the coefficient of determination (R-squared), which measures how much variance in the data set is captured by the model. R-squared → 1 as fit improves. Figure 2.8(a) shows that most benchmarks are fitted with R-squared of 0.7-1.0, indicating good fits. Benchmarks with low R-squared, such as `radiosity`, have negligible variance and no trend for Cobb-Douglas to capture.

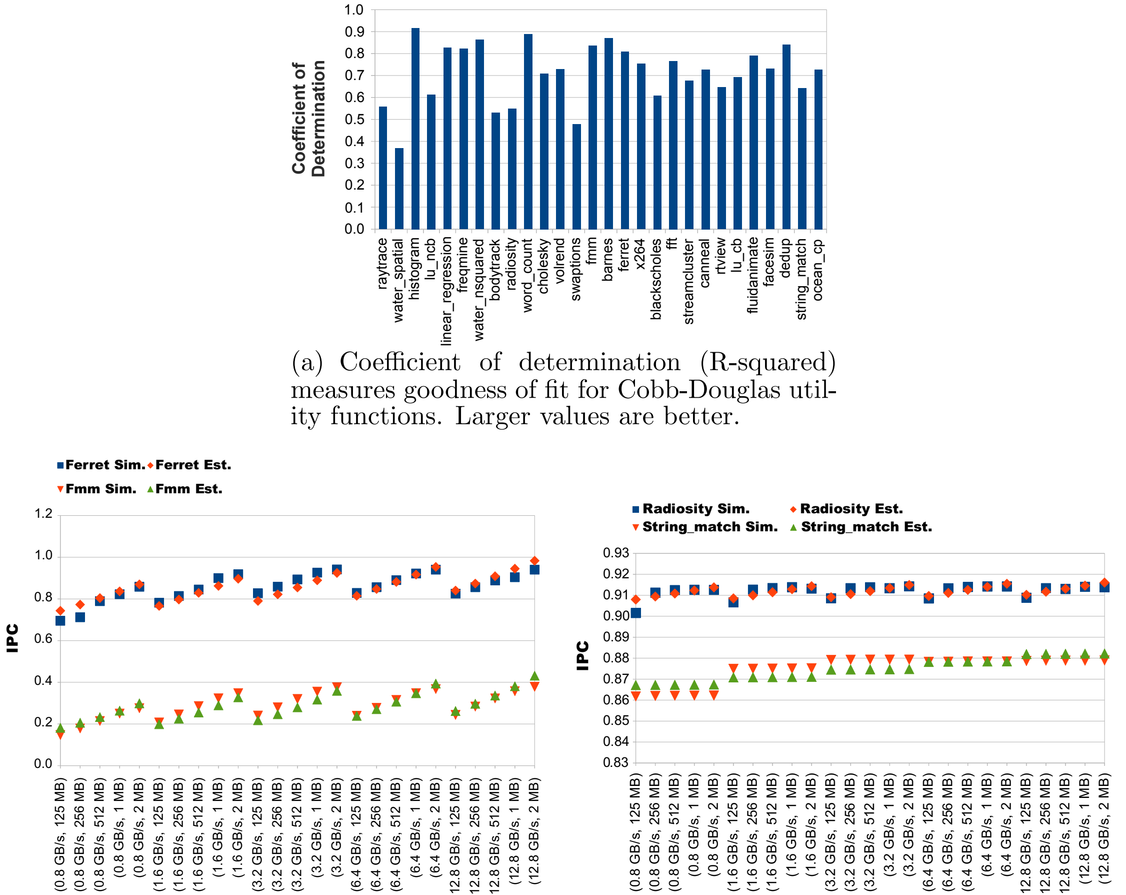
We consider representative workloads with high and low R-squared values in Figure 2.8, which plots simulated and fitted IPC. Cobb-Douglas utilities accurately track IPC and reflect preferences for cache and memory bandwidth. Even workloads with lower R-squared values, such as `radiosity`, do not deviate significantly from true values.

In practice, the proportional elasticity mechanism never uses the predicted value for u to allocate hardware. It only uses the fitted parameters for α to determine fair shares. Thus, Cobb-Douglas fits need only be good enough to assess resource elasticities and preferences. But good predictions for u give confidence in the accuracy of fitted α.

We expect Cobb-Douglas utility functions to generalize beyond cache size and memory bandwidth. After applying log transformations to performance and each of the resource allocations, our approach to fitting the utility function is equivalent to prior work in statistically inferred microarchitectural models [38]. Prior work accurately inferred performance models with more than ten microarchitectural resources, which suggests our application of Cobb-Douglas utilities will scale as more resources are shared.

2.5.3 Interpreting Cobb-Douglas Utilities

After fitting Cobb-Douglas utilities, we re-scale elasticities as described in Equation (2.12). Resource elasticity quantifies the extent to which an agent demands a resource. In other words, in a multi-resource setting, elasticities quantify the relative importance of each resource to an agent.



(a) Coefficient of determination (R-squared) measures goodness of fit for Cobb-Douglas utility functions. Larger values are better.

(b) Simulated versus fitted Cobb-Douglas performance for varied cache size, memory bandwidth allocations. Representative workloads with high R-squared.

(c) Simulated versus fitted Cobb-Douglas performance for varied cache size, memory bandwidth allocations. Representative workloads with low R-squared.

FIGURE 2.8: **Evaluating Cobb-Douglas Utilities.** Cobb-Douglas is fit by finding α with method of least squares.

Figure 2.9 depicts re-scaled elasticities for our workloads. If $\alpha_{cache} > \alpha_{mem}$, then the workload derives more utility from cache size than it does from memory bandwidth (e.g., `raytrace`). In contrast, if $\alpha_{mem} > \alpha_{cache}$, then the workload finds memory bandwidth more useful (e.g., `dedup`).

Given resource elasticities, we can classify workloads into two groups. Workloads in group M demand memory bandwidth and $\alpha_{mem} > 0.5$. Workloads in group C demand cache capacity and $\alpha_{cache} > 0.5$. This classification differentiates how workloads re-use data in their cache and whether they exhibit bursty memory behavior.

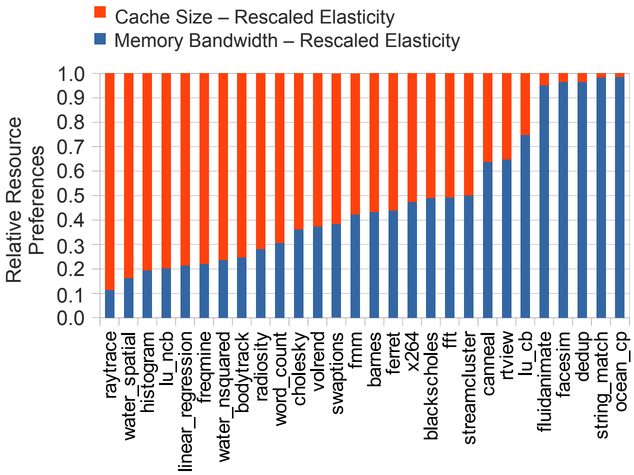


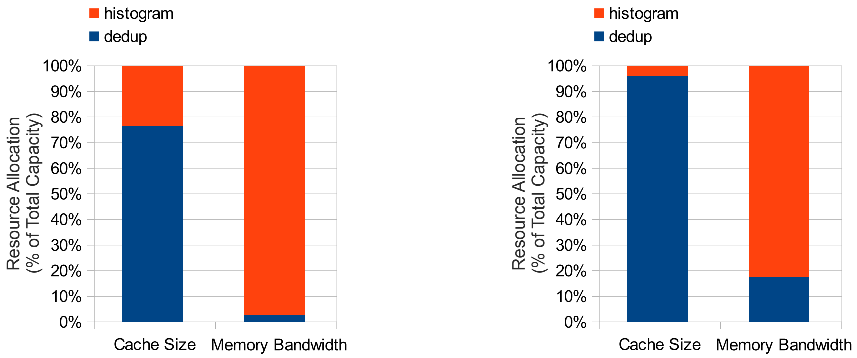
FIGURE 2.9: **Resource Preferences and Elasticities.** Re-scaled elasticities from Equation (2.12) show relative importance of cache size and memory bandwidth for each workload.

For example, `facesim`, `fluidanimate`, and `streamcluster` exhibit streaming behavior [39]. Increasing the cache size would only marginally increase performance. Streaming workloads clearly prefer memory bandwidth and this preference is reflected in their resource elasticities in Figure 2.9.

2.5.4 Proportional Elasticity versus Equal Slowdown

Having demonstrated accurate Cobb-Douglas utility models, we now evaluate our mechanism that allocates in proportion to elasticity. We compare against a mechanism that allocates for equal slowdown, a commonly used approach in computer architecture that seeks to equally distribute the performance penalties from sharing [33], [19].

We compare proportional elasticity and equal slowdown with a series of representative examples. In the first example, all desirable properties (SI, EF, PE) are satisfied by both proportional elasticity and equal slowdown. But an equal slowdown mechanism cannot guarantee these properties. We present two other examples where both SI and EF are violated by an equal slowdown mechanism.



(a) Equal Slowdown (b) Proportional Elasticity

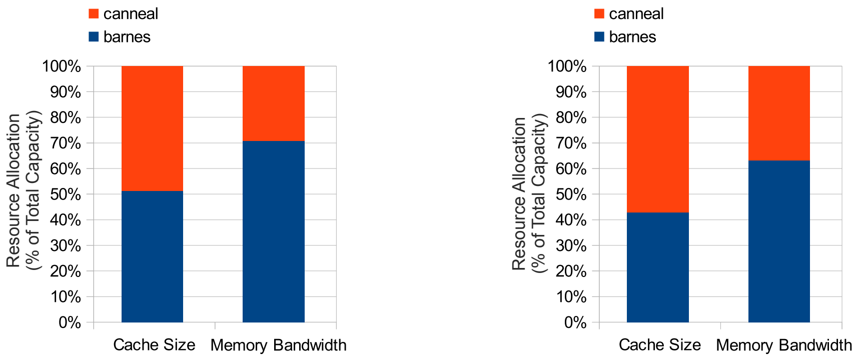
Figure 2.10: **Allocations.** Equal slowdown may satisfy SI, EF, and PE in some cases.

Example 1: C-M satisfies SI, EF, and PE. Consider a system shared by `histogram` from group C and `dedup` from group M, which prefer cache capacity and memory bandwidth, respectively. Figure 2.10 illustrates allocations as a percentage of total capacity for an equal slowdown mechanism and our proportional elasticity mechanism.

Both mechanisms allocate more cache capacity to `histogram` (C) and memory bandwidth to `dedup` (M). Consider a chip multiprocessor with 12MB cache and 24GB/s of memory bandwidth. We can compute the allocations and evaluate the conditions for SI, EF, and PE in Equation (2.11). In this particular case, the equal slowdown allocation satisfies all game-theoretic conditions for fairness. And, of course, we have proven that the proportional elasticity allocation is fair.

Unfortunately, while an equal slowdown mechanisms may provide SI and EF in this case, it cannot guarantee them. We cannot even generalize the properties of equal slowdown for broad classes of workloads. While equal slowdown happens to provide SI and EF for `histogram` (C) and `dedup` (M), it may not do so for other pairs of C and M workloads.

Example 2: C-M violates SI and EF. Figure 2.11 considers the allocations for `barnes` (C) and `canneal` (M). Barnes prefers cache size to memory bandwidth



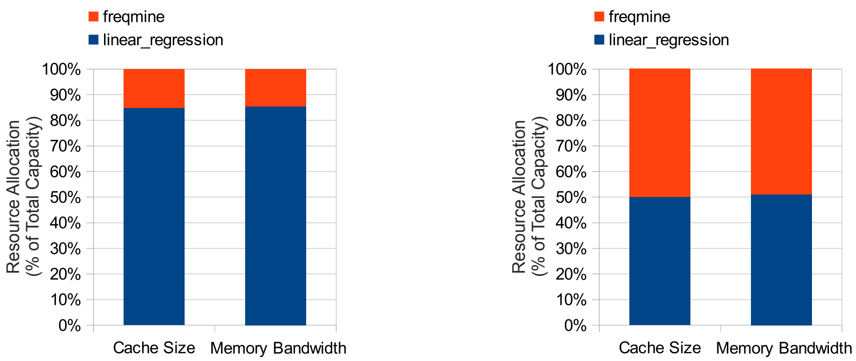
(a) Equal Slowdown (b) Proportional Elasticity

FIGURE 2.11: **Allocations.** Equal slowdown provides canneal less than half of both resources, which satisfies neither SI nor EF.

whereas cannel prefers bandwidth to cache. This example shows how an equal slowdown mechanism fails to satisfy SI and EF for `canneal`, which receives less than half of both resources in the system. In this setting, `cannel` would not be willing to participate in an equal slowdown mechanism and would rather statically receive half the hardware resources. Moreover, `canneal` envies `barnes`'s allocation. In contrast, our proportional elasticity mechanism allocates more than half of the memory bandwidth to `canneal`, giving it an incentive to share.

Example 3: C-C violates SI and EF. Finally, Figure 2.12 considers two workloads from the same group. In this, case `freqmine` (C) and `linear_regression` (C) both prefer cache capacity to memory bandwidth. But `freqmine` exhibits less memory activity than `linear`. To equalize slowdowns, `linear` must receive far more of both resources.

In this setting, `freqmine` would not be willing to share the system, preferring an equal split of the resources rather than participate in an equal slowdown mechanism. Even if `freqmine` had been willing to share resources with `linear`, it would prefer `linear`'s allocation over its own. Thus, the allocation from equal slowdown is far from equitable. On the other hand, proportional elasticity divides resources almost equally between benchmarks to satisfy SI and EF.



(a) Equal slowdown (b) Proportional elasticity

FIGURE 2.12: **Allocations.** Equal slowdown can fail to satisfy SI and EF.

However, proportional elasticity seems inefficient. It allocates resources equally when one user needs them more. Although the equal slowdown mechanism does not provide game-theoretic fairness, it likely provides higher system throughput in this example. Thus, in some cases, proportional elasticity pays a throughput penalty to provide game-theoretic fairness.

This trade-off between game-theoretic fairness and performance efficiency is fundamental to the mechanisms. The equal slowdown mechanism seeks to equalize normalized performance. If one more unit of a resource significantly improves `linear`'s performance and only modestly improve `freqmine`'s, the equal slowdown mechanism favors `linear`. And overall throughput should increase. In the next section, we quantify the performance penalty incurred by adding constraints for SI, EF, and PE.

2.5.5 Performance Penalty from Fairness

We investigate the performance lost due to game-theoretic fairness conditions. We define an agent's individual performance as $U_i(x_i) = u_i(x_i)/u_i(C)$, which divides utility when sharing by utility when not. U_i is equivalent to the notion of weighted throughput [17] except that we use utility functions rather than IPC. For each allocation policy, we compare weighted system throughput in Equation (2.17) calculated from utility functions fitted to simulator data.

- **Max Welfare w/o Fairness.** Find an allocation that maximizes welfare subject only to capacity constraints. We use Nash social welfare ($\prod_i U_i(x_i)$), which is tractably maximized with geometric programming. This mechanism provides an empirical upper bound on performance.

- **Equal Slowdown w/o Fairness.** Find an allocation that maximizes the minimum $U_i(x_i)$. This max-min objective function is equivalent to equalizing slowdown by closing the gap between the best and worst performing agents.

- **Max Welfare w/ Fairness.** Find an allocation that maximizes welfare subject to SI, EF, and PE conditions. We use Nash social welfare ($\prod_i U_i(x_i)$), which is tractably maximized with geometric programming.

- **Proportional Elasticity w/ Fairness.** Allocate in proportions based on elasticities in Cobb-Douglas utility functions. Allocations are proven to provide SI, EF, and PE.

Thus, we compare two allocations with and without game-theoretic fairness. Note that our mechanism is computationally trivial based on the closed-form expression in Equation (2.13). In contrast, the other mechanisms require geometric programming and convex optimization.

Figure 2.13 presents weighted system throughput when four applications share cache and memory bandwidth. The workloads have different characteristics, as shown in Table 2.2. Performance penalties are larger when the allocation mechanism imposes more constraints. Least restricted, maximizing welfare without any fairness constraints provides an empirical upper bound on throughput. Relative to this upper bound, equal slowdown optimizes worst-case performance, thereby low-

Table 2.2: Workload Characterization

Name	Benchmarks	C/M
WD1	histogram, linear_regression, water_nsquared, bodytrack	4C
WD2	radiosity, fmm, facesim, string_match	2C-2M
WD3	lu_cb, fluidanimate, facesim, dedup	4M
WD4	fft, streamcluster, canneal, word_count	3C-1M
WD5	streamcluster, facesim, dedup, string_match	1C-3M
WD6	histogram, linear_regression, water_nsquared, bodytrack, freqmine, word_count, x264, dedup	7C-1M
WD7	word_count, linear_regression, water_nsquared, rtview, histogram, canneal, bodytrack, radiosity	6C-2M
WD8	radiosity, word_count (2), canneal, rtview, freqmine, x264, dedup	5C-3M
WD9	radiosity (2), word_count, canneal, rtview, fmm, facesim, string_match	4C-4M
WD10	water_nsquared, barnes, ferret, lu_cb (2), fluidanimate, facesim, dedup	3C-5M

ering overall throughput. Yet, despite its lower performance, an equal slowdown mechanism does not guarantee game-theoretic fairness.

Among the two mechanisms that provide fairness with SI, EF and PE, we find no performance difference, which is a compelling result. First, our proportional elasticity

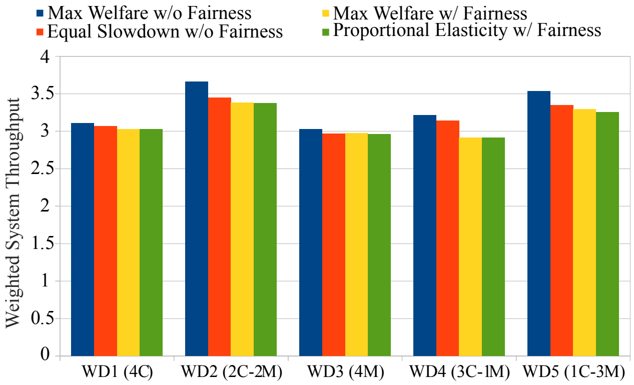


Figure 2.13: **Performance Comparison for 4-core System.** Penalties for game-theoretic fairness are less than 10%.

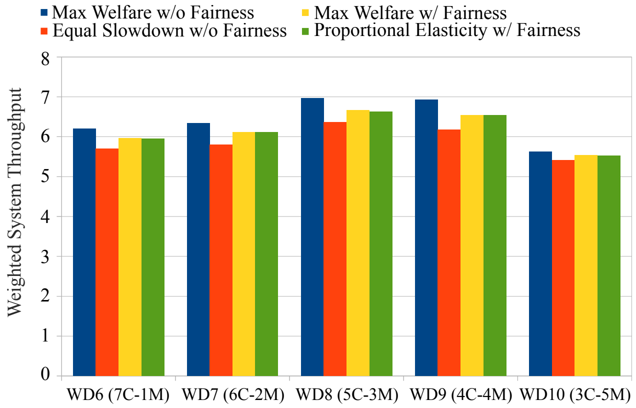


FIGURE 2.14: **Performance Comparison for 8-core System.** Penalties for game-theoretic fairness are less than 10%.

mechanism is as good as explicitly optimizing throughput subject to fairness. Second, proportional elasticity provides fair performance in a complexity effective way. Our mechanism simply calculates fair shares whereas other mechanisms would require geometric programming.

The price for game-theoretic fairness is small. First, compare maximizing welfare with and without fairness. Constraints for SI, EF, and PE reduces throughput by less than 10%. Second, compare equal slowdown to proportional elasticity. With less than a 7% throughput penalty, proportional elasticity provides game-theoretic guarantees.

Figure 2.14 further presents throughput for an eight-core system in which eight applications share cache and memory bandwidth. We select five representative workloads to compare allocation mechanisms. In this setting, constraints for game-theoretic fairness reduce throughput by less than 10%.

More interesting, in an eight-core setting, equal slowdown may perform worse than proportional elasticity. Poor performance for an equal slowdown mechanism may be due to optimizing allocations to favor the least satisfied user (i.e., max-min u_i). As the number of users increases, the opportunity cost of favoring the least satisfied user also increases. Thus, not only does an equal slowdown mechanism fail to provide game-theoretic fairness, it may also perform worse than proportional elasticity in large systems with many agents.

2.6 Related Work

Computer Science and Economics. The fair resource allocation problem has been extensively studied in computer science and economics. While most prior studies focus on fairly allocating a single resource, Ghodsi et al. propose Dominant Resource Fairness (DRF) for fair, multi-resource allocation [8]. DRF satisfies SI, PE, EF and SP for Leontief preferences. Leveraging Leontief properties, Parkes et al. [40] and Joe-Wong et al. [20] extend DRF. Dolev et al. propose an alternative notion of multi-resource fairness [21]. Gutman et al. analyze fairness frameworks and present computational tractable algorithms [41].

While Leontief preferences might be appropriate for distributed systems [8], they cannot capture important trends in hardware architecture. Leontief utilities are linear and do not allow substitution between multiple resources. Moreover, specifying a demand vector for resources, which is required by DRF, is not always possible. In this chapter, we consider fair, multi-resource allocation under Cobb-Douglas preferences, which are more realistic in computer architecture.

Fairness in Computer Architecture. Nesbit et al. [42] propose a memory scheduler to address fairness for a single memory resource. Other architects use an unfairness index [14, 18]. A variety of memory scheduling heuristics optimize this metric to fairly share memory bandwidth [43, 44, 45, 19, 33]. The unfairness index quantifies the ratio between the maximum and the minimum performance slowdown among workloads sharing the system. The allocation is considered fair if workloads experience equal slowdowns. However, we find that equal slowdowns cannot guarantee game-theoretic properties (e.g., SI, EF, PE).

We consider fairness in a multi-resource setting. Coordinating multi-resource allocation is more challenging due to substitution effects. Bitirgen et al. [19] consider multiple resources, relying on machine learning to predict performance for different allocations at run-time. Their objective is system throughput not fairness. Moreover, their learning technique is likely more computationally demanding than our equation for fair shares.

Resource Allocation in Datacenters. Within datacenters, market mechanisms allocate resources to maximize welfare, which is defined as user utility minus power cost [46], [47].

2.7 Conclusions

Our results motivate new thinking in fairly allocating hardware resources. Rather than assume users must share hardware, we must provide allocation mechanisms to encourage sharing. We show that Cobb-Douglas utilities are well suited to modeling user preferences in computer architecture. For Cobb-Douglas utilities, we present an allocation mechanism that provides sharing incentives, envy-freeness, Pareto efficiency, and strategy-proofness in the large. By linking hardware resource management to robust, game-theoretic analysis, computer architects can qualitatively change the nature of performance guarantees in hardware platforms shared by strategic users.

3

The Computational Sprinting Game

3.1 Introduction

Modern datacenters oversubscribe their power supplies to enhance performance and efficiency. A conservative datacenter that deploys servers according to their expected power draw will under-utilize provisioned power, operate power supplies at sub-optimal loads, and forgo opportunities for higher performance. In contrast, efficient datacenters deploy more servers than it can power fully and rely on varying computational load across servers to modulate demand for power [48]. Such a strategy requires responsive mechanisms for delivering power to the computation that needs it most.

Computational sprinting is a class of mechanisms that supply additional power for short durations to enhance performance. In chip multiprocessors, for example, sprints activate additional cores and boost their voltage and frequency. Although originally proposed for mobile systems [49, 50], sprinting has found numerous applications in datacenter systems. It can accelerate computation for complex tasks or accommodate transient activity spikes [51, 52].

The system architecture determines sprint duration and frequency. Sprinting multiprocessors generate extra heat, absorbed by thermal packages and phase change materials [49, 52], and require time to release this heat between sprints. At scale, uncoordinated multiprocessors that sprint simultaneously could overwhelm a rack or cluster's power supply. Uninterruptible power supplies reduce the risk of tripping circuit breakers and triggering power emergencies. But the system requires time to recharge batteries between sprints. Given these physical constraints in chip multiprocessors and the datacenter rack, sprinters require recovery time. Thus, sprinting mechanisms couple performance opportunities with management constraints.

We face fundamental management questions when servers sprint independently but share a power supply – which processors should sprint and when should they sprint? Each processor's workload derives extra performance from sprinting that depends on its computational phase. Ideally, sprinters would be the processors that benefit most from boosted capability at any given time. Moreover, the number of sprinters would be small enough to avoid power emergencies, which constrain future sprints. Policies that achieve these goals are prerequisites for sprinting to full advantage.

We present the computational sprinting game to manage a collection of sprinters. The sprinting architecture, which defines the sprinting mechanism as well as power and cooling constraints, determines rules of the game. A strategic agent, representing a multiprocessor and its workload, independently decides whether to sprint at the beginning of an epoch. The agent anticipates her action's outcomes, knowing that the chip must cool before sprinting again. Moreover, she analyzes system dynamics, accounting for competitors' decisions and risk of power emergencies.

We find the equilibrium in the computational sprinting game, which permits distributed management. In an equilibrium, no agent can benefit by deviating from her optimal strategy. The datacenter relies on agents' incentives to decentralize management as each agent self-enforces her part of the sprinting policy. Decentralized equilibria allow datacenters to avoid high communication costs and unwieldy enforcement mechanisms in centralized management. Moreover, equilibria outperform prior heuristics. In summary, we present the following contributions:

- **Sprinting Architecture (Section 3.2).** We present a system of independent sprinters that share power – a rack of chip multiprocessors. Sprinting multiprocessors activate additional cores and increase clock rates. Sprints are constrained by chips' thermal limits and rack power limits.

- **Sprinting Game (Section 3.3).** We define a repeated game in which strategic agents sprint based on application phases and system conditions. The game divides time into epochs and agents play repeatedly. Actions in the present affect performance and the ability to sprint in the future.

- **Dynamics and Strategies (Section 3.4).** We design agents who sprint when the expected utility from doing so exceeds a threshold. We devise an algorithm that optimizes each agent's threshold strategy. The strategies produce an equilibrium in which no agent benefits by deviating from her optimal threshold.

- **Performance (Sections 3.5–3.6).** We evaluate the game for Spark-based datacenter applications, which exhibit diversity in phase behavior and utility from sprinting. The game increases task throughput by 4-6× when compared to prior heuristics in which agents sprint greedily.

3.2 The Sprinting Architecture

We present a sprinting architecture for chip multiprocessors in datacenters. Multiprocessors sprint by activating additional cores and increasing their voltage and frequency. Datacenter applications, with their abundant task parallelism, scale across

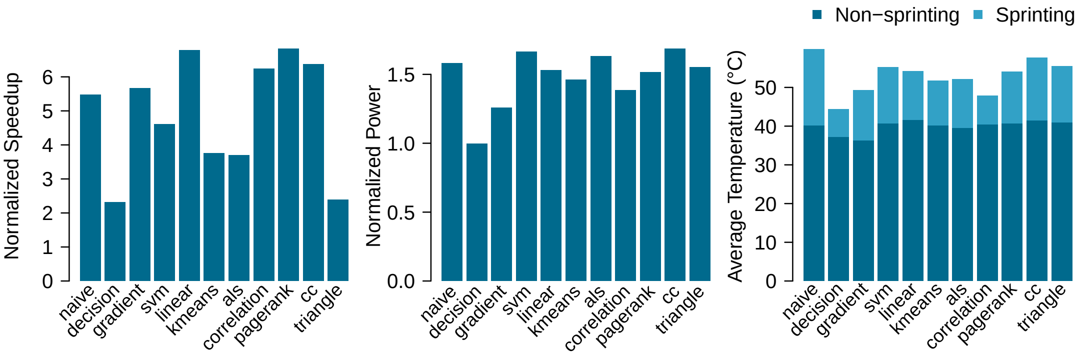


FIGURE 3.1: **Normalized Speedup, Power, and Temperature.** Shown for varied Spark benchmarks when sprinting. Nominal operation supplies three cores at 1.2GHz. Sprint supplies twelve cores at 2.7GHz.

additional cores as they become available. We focus on applications built atop the Spark framework, which extends Hadoop for memory caching [53]. In Figure 3.1, Spark benchmarks perform 2-7× better on a sprinting multiprocessor, but dissipates 1.8× the power. Power produces heat.

Sprinters require infrastructure to manage heat and power. First, the chip multiprocessor's thermal package and heat sink must absorb surplus heat during a sprint [49, 54]. Second, the datacenter rack must employ batteries to guard against power emergencies caused by a surplus of sprinters on a shared power supply. Third, the system must implement management policies that determine which chips sprint.

3.2.1 Chip Multiprocessor Support

A chip multiprocessor's maximum power level depends on its thermal package and heat sink. Given conventional heat sinks, thermal constraints are the primary determinant of multiprocessor performance, throttling throughput and overriding constraints from power delivery and off-chip bandwidth [55]. More expensive heat sinks employ phase change materials (PCMs), which increase thermal capacitance, to absorb and dissipate excess heat [50, 54]. The quality of the thermal package, as measured by its thermal capacitance and conductance, determines parameters of the sprinting game.

The choice of thermal package dictates the maximum duration of a sprint [54]. Whereas water, air and foam enable sprint durations on the order of seconds [50], PCMs enable durations on the order of minutes if not hours [56, 57, 54]. Our sprint architecture employs paraffin wax, which is attractive for its high thermal capacitance and tunable melting point when blended with polyolefins [58]. We estimate a chip with paraffin wax can sprint with durations on the order of 150 seconds.

After a sprint, the thermal package must release its heat before the chip can sprint again. The average cooling duration, denoted as Δt_{cool}, is the time required before the PCM returns to ambient temperature. The rate at which the PCM dissipates heat depends on its melting point and the thermal resistance between the material and the ambient [58]. Both factors can be engineered and, with paraffin wax, we estimate a cooling duration on the order of 300 seconds, twice the sprint's duration.

Different types of workloads may demand different sprint durations. Sprints for online queries requires tens of milliseconds or less [59]. Sprints for parallel workloads requires seconds or more [50]. And those for warehouse-scale thermal management requires support for hours [52]. In this chapter, we study data analytics applications that would prefer to sprint indefinitely. In this setting, the primary determinant of a sprint's duration is the thermal package.

3.2.2 Datacenter Support

At scale, servers within the same rack share a power supply. Chip multiprocessors draw current from a shared power distribution unit (PDU) that is connected to a branch circuit and protected by a circuit breaker (CB). Datacenter architects deploy servers to oversubscribe branch circuits for efficiency. Oversubscription utilizes a larger fraction of the facility's provisioned power for computation. But it relies on power capping and varied computational load across servers to avoid tripping circuit breakers or violating contracts with utility providers [48, 60]. Although sprints boost computation for complex queries and during peak loads [59, 51], the risk of a power emergency increases with the number of sprinters in a power capped datacenter.

Circuit Breakers and Trip Curves. Figure 3.2 presents the circuit breaker's trip curve, which specifies how sprint duration and power combine to determine whether the breaker trips. The trip time corresponds to the sprint's duration. Longer sprints increase the probability of tripping the breaker. The current draw corresponds to the number of simultaneous sprints as each sprinter contributes to the load above rated current. Higher currents increase the probability of tripping the breaker. Thus, the tolerance for sprints depends on their duration and power. The breaker dictates the number of sprinters supported by the datacenter rack.

Figure 3.3 associates the number of sprinters to the tripping probability for a given trip time. Let n_S denote the number of sprinters and let P_{trip} denote the probability of tripping the breaker. The breaker occupies one of the following regions:

- **Non-Tripped.** P_{trip} is zero when $n_S < N_{\min}$
- **Non-Deterministic.** P_{trip} is a non-decreasing function of n_S when $N_{\min} \leq n_S < N_{\max}$
- **Tripped.** P_{trip} is one when $n_S \geq N_{\max}$

Note that $N_{\min}$ and $N_{\max}$ depend on the breaker's trip curve and the application's demand for power when sprinting.

Suppose a sprinter dissipates twice as much power as a non-sprinter, as in Spark applications on chip multiprocessors. We find that the breaker does not trip when less than 25% of the chips sprint and definitely trips when more than 75% of the chips sprint. In other words, $N_{\min} = 0.25N$ and $N_{\max} = 0.75N$. We consider UL489 circuit breakers from Rockwell Automation, which can be overloaded to 125-175% of rated current for a 150 second sprint [51, 61, 62].

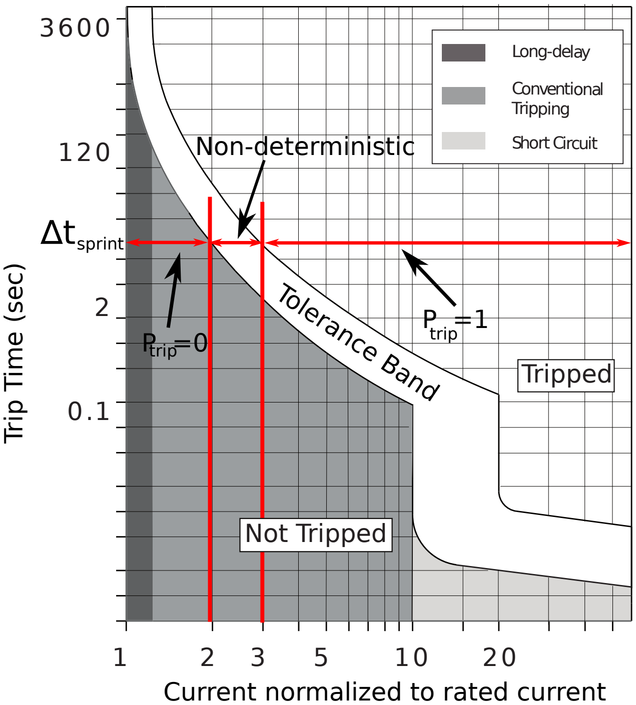


FIGURE 3.2: **Typical Trip Curve of a Circuit Breaker.** [60].

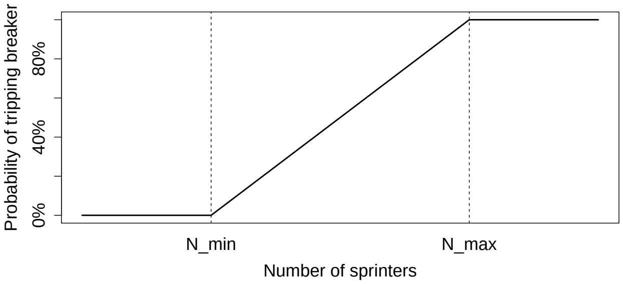


FIGURE 3.3: **Probability of Tripping the Rack's Circuit Breaker.**

Uninterruptible Power Supplies. When the breaker trips and resets, power distribution switches from the branch circuit to the uninterruptible power supply (UPS) [63, 64]. The rack augments power delivery with batteries to complete sprints in progress. Lead acid batteries support discharge times of 5-120 minutes, long enough to support the duration of a sprint. After completing sprints and resetting the breaker, servers resume computation on the branch circuit.

However, servers are forbidden from sprinting again until UPS batteries have been recharged. Sprints before recovery would compromise server availability and

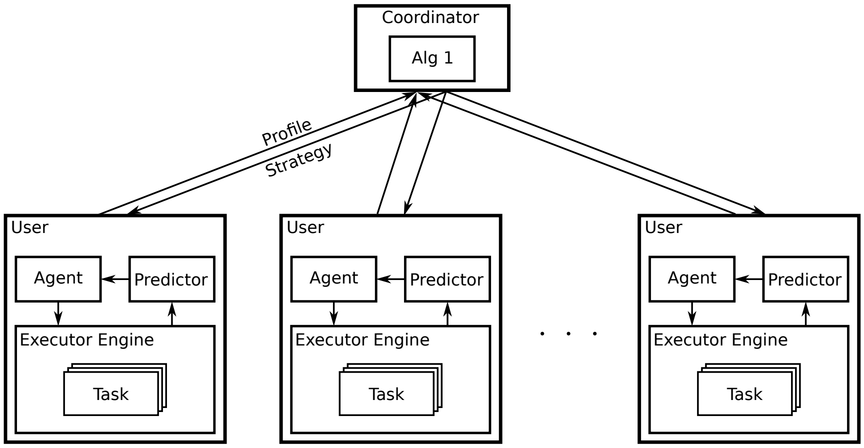


FIGURE 3.4: **Sprinting Architecture.** Users deploy task executors and agents that decide when to sprint. Agents send performance profiles to a coordinator and receives optimized sprinting strategies.

increase vulnerability to power emergencies. Moreover, frequent discharges without recharges would shorten battery life. The average recovery duration, denoted by $\Delta t_{\text{recover}}$, depends on the UPS discharge depth and recharging time. A battery can be recharged to 85% capacity in 8-10× the discharge time [65], which corresponds to 8-10× the sprint duration.

Servers are permitted to sprint again after recharge and recovery. However, if every chip multiprocessor in the rack were to sprint simultaneously and immediately after recovery, they would trigger another power emergency. The rack must stagger the distribution of sprinting permissions to avoid dI/dt problems.

3.2.3 Power Management

Figure 3.4 illustrates the management framework for a rack of sprinting chip multiprocessors. The framework supports policies that pursue the performance of sprints while avoiding system instability. Unmanaged and excessive sprints may trip breakers, trigger emergencies, and degrade performance at scale. The framework achieves its objectives with strategic agents and coarse-grained coordination.

Users and Agents. Each user deploys three run-time components: executor, agent, and predictor. Executors provide clean abstractions, encapsulating applications that could employ different software frameworks [66]. The executor supports task-parallel computation by dividing an application into tasks, constructing a task dependence graph, and scheduling tasks dynamically based on available resources. Task scheduling is particularly important as it increases parallelism when sprinting powers-on cores and tolerates faults when cooling and recovery powers-off cores.

Agents are strategic and selfish entities that act on users' behalf. They decide whether to sprint by continuously analyzing fine-grained application phases. Because sprints are followed by cooling and recovery, an agent sprints judiciously and targets application phases that benefit most from extra capability. Agents use predictors that estimate utility from sprinting based on software profiles and hardware counters. Each agent represents a user and her application on a chip multiprocessor.

Coordination. The coordinator collects profiles from all agents and assigns tailored sprinting strategies to each agent. The coordinator interfaces with strategic agents who may attempt to manipulate system outcomes by misreporting profiles or deviating from assigned strategies. Fortunately, our game-theoretic mechanism guards against such behavior.

First, agents will truthfully report their performance profiles. In large systems, game theory provides incentive compatibility, which means that agents cannot improve their utility by misreporting their preferences. The coordinator assigned a tailored strategy to each agent based on system conditions. An agent who misreports her profile has little influence on conditions in a large system. Not only does she fail to affect others, an agent who misreports suffers degraded performance as the coordinator assigns her a poorly suited strategy based on inaccurate profiles.

Second, agents will implement their assigned strategies because the coordinator optimizes those strategies to produce an equilibrium. In equilibrium, every agent implements her strategy and no agent benefits when deviating from it. An equilibrium has compelling implications for management overheads. If each agent knows that every other agent is playing her assigned strategy, she will do the same without further communication with the coordinator. Global communication between agents and the coordinator is infrequent and occurs only when system profiles change. Local communication between each user's run-time components (*i.e.*, executor, agent, predictor) is frequent but employs inexpensive, inter-process mechanisms. In effect, an equilibrium permits the distributed enforcement of sprinting policies.

In contrast, the centralized enforcement of coordinated policies poses several challenges. First, it requires frequent and global communication as each agent decides whether to sprint by querying the coordinator at the start of each epoch. The length of an epoch is short and corresponds to sprint duration. Moreover, without equilibria, agents with kernel privileges could ignore prescribed policies, sprint at will, and cause power emergencies that harm all agents. Avoiding such outcomes in a multi-tenant datacenter would require a distributed runtime. The runtime, not the agent, would have kernel privileges for power management, introducing an abstraction layer and overheads.

3.3 The Sprinting Game

We present a computational sprinting game, which governs demands for power and manages system dynamics. We design a dynamic game that divides time into epochs and asks agents to play repeatedly. Agents represent chip multiprocessors that share a power supply. Each agent chooses to sprint independently, pursuing benefits in the current epoch and estimating repercussions in future epochs. Multiple agents can sprint simultaneously, but they risk tripping the circuit breaker and triggering power emergencies that harm global performance.

3.3.1 Game Formulation

The game considers N agents who run task-parallel applications on N chip multiprocessors. Each agent computes in either normal or sprinting mode. The normal mode uses a fraction of the cores at low frequency whereas sprints use all cores at high frequency. Sprints rely on the executor to increase task parallelism and exploit extra cores. In this chapter, for example, we consider three cores at 1.2GHz in normal mode and twelve cores at 2.7GHz in a sprint.

The repeated game divides time into epochs. The duration of an epoch corresponds to the duration of a safe sprint, which neither overheats the chip nor trips the circuit breaker. An agent's utility from a sprint varies across epochs according to her application's phases. Agents apply a discount factor $\delta < 1$ to future utilities as, all else being equal, they prefer performance sooner rather than later.

3.3.2 Agent States

At any given time, an agent occupies one of three states—active (A), chip cooling (C), and rack recovery (R)—according to her actions and those of others in the rack. An agent's state describes whether she can sprint, and describes how cooling and recovery impose constraints on her actions.

Active (A) – Agent can safely sprint. By default, an agent in an active state operates her chip in normal mode, with a few processor cores running at low frequency. The agent has an option to sprint, which deploys additional cores and raises the frequency. She decides whether to sprint by comparing a sprint's benefits in the current epoch against benefits from deferring the sprint to a future epoch. If the agent sprints, her state in the next epoch is cooling.

Chip Cooling (C) – Agent cannot sprint. After a sprint, an agent remains in the cooling state until excess heat has been dissipated. Cooling requires a number of epochs Δt_{cool}, which depends on the chip's thermal conductance and resistance, the heat sink and cooling technology, and the ambient temperature. An agent in the cooling state stays in this state with probability p_c and returns to the active state with probability $1 - p_c$. Probability p_c is defined so that $1/(1 - p_c) = \Delta t_{\text{cool}}$.

Rack Recovery (R) – Agent cannot sprint. When multiple chips sprint simultaneously, their total current draw may trip the rack's circuit breaker, trigger a power emergency, and require supplemental current from batteries. After an emergency, all agents remain in the recovery state until batteries recharge. Recovery requires a number of epochs $\Delta t_{\text{recover}}$, which depends on the rack's power supply and its battery capacity. Agents in the recovery state stay in this state with probability p_r and return to the active state with probability $1 - p_r$. Probability p_r is defined so that $1/(1 - p_r) = \Delta t_{\text{recover}}$.

In summary, the states describe and enforce system constraints. A chip that sprints must cool before sprinting again. A rack that supports sprints with batteries must recharge those batteries before doing so again. Agents in cooling or recovery states are constrained, but those in active states will sprint strategically.

3.3.3 Agent Actions and Strategies

Agents have two possible actions — sprint or do not sprint. Strategic agents decide between these actions to maximize their utilities. Each agent's sprinting strategy depends on various factors, including

- agent's state and her utility from sprinting,
- agent's history of sprinting,
- other agents' states,
- other agents' utilities, strategies, and histories.

Sprinting strategies determine the game's performance. Agents that greedily sprint at every opportunity produce several sub-optimal outcomes. First, chips and racks would spend many epochs in cooling and recovery states, respectively, degrading system throughput. Moreover, agents who sprint at the first opportunity constrain themselves in future epochs, during which sprints may be even more beneficial.

In contrast, sophisticated strategies improve agent utility and system performance. Strategic agents sprint during the epochs that benefit most from additional cores and higher frequencies. Moreover, they consider other agents' strategies because the probability of triggering a power emergency and entering the recovery state increases with the number of sprinters. We analyze the game's governing dynamics to optimize each agent's strategy and maximize her performance.

3.4 Game Dynamics and Agent Strategies

A comprehensive approach to optimizing strategies considers each agent—her state, utility, and history—to determine whether sprinting maximizes her performance given her competitor's strategies and system state. In practice, however, this optimization does not scale to hundreds or thousands of agents.

For tractability, we analyze the population of agents by defining key probability distributions on population behavior. This approach has several dimensions. First, we reason about population dynamics in expectation and consider an "average" agent. Second, we optimize each agent's strategy in response to the population rather than individual competitors. Third, we find an equilibrium in which no agent can perform better by deviating from her optimal strategy.

3.4.1 Mean Field Equilibrium

The mean field equilibrium (MFE), a concept drawn from economic game theory, is an approximation method used when analyzing individual agents in a large system is intractable [67, 68, 69, 70, 71, 72]. With the MFE, we can characterize expected behavior for a population of agents and then optimize each agent's strategy against that expectation. We can reason about the population and neglect individual agents because any one agent has little impact on overall behavior in a large system.

The mean field analysis for the sprinting game focuses on the sprint distribution, which characterizes the number of agents who sprint when the rack is not in the recovery state. In equilibrium, the sprint distribution is stationary and does not change across epochs. In any given epoch, some agents complete a sprint and enter the cooling state while others leave the cooling state and begin a sprint. Yet the number of agents who sprint is unchanged in expectation.

The stationary distribution for the number of sprinters translates into stationary distributions for the rack's current draw and the probability of tripping the circuit breaker – see Figure 3.3. Given the rack's tripping probability, which concisely describes population dynamics, an agent can formulate her best response and optimize her sprinting strategy to maximize performance.

We find an equilibrium by characterizing a population's statistical distributions, optimizing agents' responses, and simulating game play to update the population. We specify an initial value for the probability of tripping the breaker and iterate as follows.

- **Optimize Sprint Strategy (Section 3.4.2).** Given the probability of tripping the breaker P_{trip}, each agent optimizes her sprinting strategy to maximize her performance. She sprints if performance gains from doing so exceed some threshold. Optimizing her strategy means setting her threshold u_T.

- **Characterize Sprint Distribution (Section 3.4.3).** Given that each agent sprints according to her threshold u_T, the game characterizes population behavior. It estimates the expected number of sprinters n_S, calculates their demand for power, and updates the probability of tripping the breaker P'_{trip}.

- **Check for Equilibrium.** The game is in equilibrium if $P'_{\text{trip}} = P_{\text{trip}}$. Otherwise, iterate with the new probability of tripping the breaker.

3.4.2 Optimizing the Sprint Strategy

An agent considers three factors when optimizing her sprinting strategy: the probability of tripping the circuit breaker P_{trip}, her utility from sprinting u, and her state. An agent occupies either the active (A), cooling (C), or recovery (R) state. To maximize expected value and decide whether to sprint, each agent optimizes the following Bellman equation.

$$V(u,\ A) = \max\{V_S(u,\ A),\ \ V_{\neg S}(u,\ A)\} \tag{3.1}$$

The Bellman equation quantifies value when an agent acts optimally in every epoch. V_S and $V_{\neg S}$ are the expected values from sprinting and not sprinting, respectively. If $V_S(u,\ A) > V_{\neg S}(u,\ A)$, then sprinting is optimal. The game solves the Bellman equation and identifies actions that maximize value with dynamic programming.

Value in Active State. Sprinting defines a repeated game in which an agent acts in the current epoch and encounters consequences of that action in future epochs. Accordingly, the Bellman equation is recursive and expresses an action's value in terms of benefits in the current epoch plus the discounted value from future epochs.

Suppose an agent in the active state decides to sprint. Her value from sprinting is her immediate utility u plus her discounted utility from future epochs. When she sprints, her future utility is calculated for the chip cooling state $V(C)$ or calculated for the rack recovery state $V(R)$ when her sprint trips the circuit breaker.

$$V_S(u,\ A) = u + \delta\left[V(C)(1 - P_{\text{trip}}) + V(R)P_{\text{trip}}\right] \tag{3.2}$$

On the other hand, an agent who does not sprint will remain in the active state unless other sprinting agents trip the circuit breaker and trigger a power emergency that requires recovery.

$$V_{\neg S}(u,\ A) = \delta\left[V(A)(1 - P_{\text{trip}}) + V(R)P_{\text{trip}}\right] \tag{3.3}$$

We use $V(A)$ to denote an agent's expected value from being in the active state, which depends on an agent's utility from sprinting. The game profiles an application and its time-varying computational phases to obtain a probability density function $f(u)$, which characterizes how often an agent derives utility u from sprinting. With this density, the game estimates expected value.

$$V(A) = \int V(u,\ A)\ f(u)\ du \tag{3.4}$$

Value in Cooling and Recovery States. An active agent transitions into cooling and recovery states when she and/or others sprint. Because agents cannot sprint while cooling or recovering, their expected values from these states do not depend on their utility from sprinting.

$$\begin{aligned} V(C) = &\ \ \delta\ \left[V(C)p_c + V(A)(1-p_c)\right](1 - P_{\text{trip}}) + \\ &\ \ \delta\ V(R)P_{\text{trip}} \qquad (3.5) \\ V(R) = &\ \ \delta\ \left[V(R)p_r + V(A)(1-p_r)\right] \qquad (3.6) \end{aligned}$$

Parameters p_c and p_r are technology-specific probabilities of an agent in cooling and recovery states staying in those states. An agent in cooling will remain in this state with probability p_c and become active with probability $1 - p_c$, assuming the rack avoids a power emergency. If the circuit breaker trips, an agent enters recovery. An agent remains in recovery with probability p_r and becomes active with probability $1 - p_r$. The game tunes these parameters to reflect the time required for chip cooling after a sprint and for rack recovery after a power emergency – see Section 3.2.

Threshold Strategy. An agent should sprint if her utility from doing so is greater than not. But when is this the case? Equation (3.8), which follows from

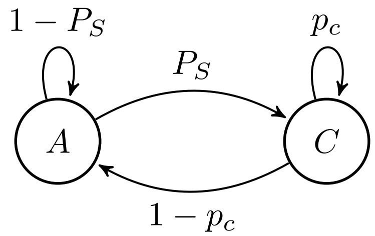


FIGURE 3.5: **State Transitions.** When agent sprints, chip cools. Assumes an agent is not in recovery.

Equations (3.2)–(3.3), states that an agent should sprint if her utility u is greater than her optimal threshold for sprinting u_T.

$$V_S(u,\ A) \ > \ V_{\neg S}(u,\ A) \tag{3.7}$$

$$u \ > \ \underbrace{\delta\ (V(A) - V(C))\ (1 - P_{\text{trip}})}_{u_T} \tag{3.8}$$

Thus, an agent uses threshold u_T to test a sprint's utility. If sprinting improves performance by more than the threshold, an agent should sprint. Applying this strategy in every epoch maximizes expected value across time in the repeated game.

3.4.3 Characterizing the Sprint Distribution

Given threshold u_T for her strategy, an agent uses her density function on utility to estimate the probability that she sprints, p_s, in a given epoch.

$$p_s = \int_{u_T}^{u_{\max}} f(u)\ du \tag{3.9}$$

The probabilities of sprinting (p_s) and cooling (p_c) define a Markov chain that describes each agent's behavior – see Figure 3.5, which assumes the agent is not in recovery. As agents play their strategies, the Markov chain converges to a stationary distribution in which each agent is active with probability p_A. If N agents play the game, the expected number of sprinters is

$$n_S = p_s \times p_A \times N \tag{3.10}$$

Algorithm 1: Optimizing the Sprint Strategy

```
input  : Probability density function for sprinting utilities ($f(u)$)
output: Optimal sprinting threshold ($u_T$)
$j \leftarrow 1$
$P^0_{\text{trip}} \leftarrow 1$
while $P^j_{trip}$ not converged do
    $u^j_T \leftarrow$ DP solution for Equations (3.1)–(3.8) with $P^j_{\text{trip}}$
    $p^j_S \leftarrow$ Equation (3.9) with $f(u)$, $u^j_T$
    $n^j_S \leftarrow$ Equation (3.10) with MC solution and $P^j_S$
    $P^{j+1}_{\text{trip}} \leftarrow$ Equation (3.11)
    $j \leftarrow j + 1$
end
```

As the number of sprinters increases, so does the rack's current draw and the probability of tripping the breaker. Given the expected number of sprinters, the game updates the probability of tripping the breaker according to its trip curve (*e.g.*, Figure 3.3). Mathematically, the curve is described as follows.

$$P_{\text{trip}} = \begin{cases} 0 & \text{if } n_S < N_{\min} \\ \frac{n_S - N_{\min}}{N_{\max} - N_{\min}} & \text{if } N_{\min} \leq n_S \leq N_{\max} \\ 1 & \text{if } n_S > N_{\max} \end{cases} \tag{3.11}$$

P_{trip} determines n_S, which determines P'_{trip}. If $P_{\text{trip}} = P'_{\text{trip}}$, then agents are playing optimized strategies that produce an equilibrium.

3.4.4 Finding the Equilibrium

When the game begins, agents make initial assumptions about population behavior and the probability of tripping the breaker. Agents optimize their strategies in response to population behavior. Strategies produce sprints that affect the probability of tripping the breaker. Over time, population behavior and agent strategies converge to a stationary distribution, which is consistent across epochs. The game is in equilibrium if the following conditions hold.

- Given tripping probability P_{trip}, the sprinting strategy dictated by threshold

Table 3.1: Spark Workloads

Benchmark	Category	Dataset	Data Size
NaiveBayesian	Classification	kdda2010 [73]	2.5G
DecisionTree	Classification	kdda2010	2.5G
GradientBoostedTrees	Classification	kddb2010 [73]	4.8G
SVM	Classification	kdda2010	2.5G
LinearRegression	Classification	kddb2010	4.8G
Kmeans	Clustering	uscensus1990 [74]	327M
ALS	Collaborative Filtering	movielens2015 [75]	325M
Correlation	Statistics	kdda2010	2.5G
PageRank	Graph Processing	wdc2012 [76]	5.3G
ConnectedComponents	Graph Processing	wdc2012	5.3G
TriangleCounting	Graph Processing	wdc2012	5.3G

u_T is optimal and solves the Bellman equation in Equations (3.1)–(3.3).

- Given sprinting strategy u_T, the probability of tripping the circuit breaker is P_{trip} and is calculated by Equations (3.9)–(3.11).

In equilibrium, every agent plays her optimal strategy and no agent benefits when deviating from her strategy. In practice, the coordinator in the management framework finds and maintains an equilibrium with a mix of offline and online analysis.

Offline Analysis. Agents sample epochs and measure utility from sprinting to produce a density function $f(u)$, which characterizes how often an agent sees utility u from sprinting. The coordinator collects agents' density functions, analyzes population dynamics, and tailors sprinting strategies for each agent. Finally, the coordinator assigns optimized strategies to support online sprinting decisions.

Algorithm 1 describes the coordinator's offline analysis. It initializes the probability of tripping the breaker. Then it iteratively analyzes population dynamics to find an equilibrium. Each iteration proceeds in three steps. First, the coordinator optimizes sprinting threshold u_T by solving the dynamic program defined in Equations (3.1)–(3.8). Second, it estimates the number of sprinters according to Equation

(3.10). Finally, it updates the probability of tripping the breaker according to Equation (3.11). The algorithm terminates when thresholds, number of sprinters, and tripping probability are stationary.

The offline algorithm has no performance overhead. The analysis runs periodically to update sprinting strategies and the tripping probability as application mix and system conditions evolve. It does not affect an application's critical path as agents use updated strategies when they become available but need not wait for them.

The algorithm requires little computation. It solves the dynamic program with value-iteration, which has a convergence rate that depends on the discount factor δ. The number of iterations grows polynomially in $(1-\delta)^{-1}$. We implement and run the algorithm on an Intel® Core™ i5 processor with 4GB of memory. The algorithm completes in less than 10s, on average.

Online Strategy. An agent decides whether to sprint at the start of each epoch by estimating a sprint's utility and comparing it against her threshold. Estimation could be implemented in several ways. An agent could use the first few seconds of an epoch to profile her normal and sprinting performance. Alternatively, an agent could use heuristics to estimate utility from additional cores and higher clock rates. For example, task queue occupancy and cache misses are associated with a sprint's impact on task parallelism and instruction throughput, respectively. Comparisons with a threshold are trivial. If an agent decides to sprint, it turns on otherwise disabled cores using CPU-hotplug and increases clock rates using ACPI [77].

3.5 Experimental Methodology

Servers and Sprints. The agent and its application are pinned to a chip multiprocessor, an Intel® Xeon® E5-2697 v2 that can run at 2.70GHz. Two multiprocessors share 128GB of main memory within a server. An agent runs in normal or sprinting mode. In normal mode, the agent uses three 1.2GHz cores. In sprinting mode, the agent uses twelve 2.7GHz cores. We turn cores on and off with Linux `sysfs`. In principle, sprinting represents any mechanism that performs better but consumes more power.

Workloads. We evaluate Apache Spark workloads [53]. The Spark run-time engine dynamically schedules tasks to use available cores and maximize parallelism, adapting as sprints cause the number of available cores to vary across epochs. Each agent runs a Spark application on representative datasets as shown in Table 3.1.

Profiling Methods. We collect system profiles that measure power and temperature, using the Intel® Performance Counter Monitor 2.8 to read MSR registers once every second. We collect workload profiles by modifying Spark (v1.3.1) to log the IDs of jobs, stages, and tasks upon their completion.

We measure application performance in terms of the number of tasks completed per second (TPS). Each application defines a number of jobs, and each job is divided into tasks that compute in parallel. Jobs are completed in sequence while tasks can be completed out of order. The total number of tasks in a job is constant and independent of the available hardware resources. Thus, TPS measures performance for a fixed amount of work.

We trace TPS during an application's end-to-end execution in normal and sprinting modes. Since execution times differ in the two modes, comparing traces requires some effort. For every second in normal mode, we measure the number of tasks completed and estimate the number of tasks that would have been completed in the sprinting mode. For our evaluation, we estimate a sprint's speedup by comparing the measured non-sprinting trace and the interpolated sprinting trace. In a practical system, online profiling and heuristics would be required.

Simulation Methods. We simulate 1000 users and evaluate their performance in the sprinting game. The R-based simulator uses traces of Spark computation col-

Table 3.2: Experimental Parameters

Description	Symbol	Value
Min # sprinters	N_{min}	250
Max # sprinters	N_{max}	750
Prob. of staying in cooling	p_c	0.50
Prob. of staying in recovery	p_r	0.88
Discount factor	δ	0.99

lected in both normal and sprinting modes. The simulator models system dynamics as agents sprint, cool, and recover.

One set of simulations evaluates homogeneous agents who arrive randomly and launch the same type of Spark application; randomized arrivals cause application phases to overlap in diverse ways. A second set of simulations evaluates heterogeneous agents who launch different types of applications, further increasing the diversity of overlapping phases. Diverse phase behavior exercises the sprinting game as agents and their processors optimize strategies in response to varied competitors'.

Table 3.2 summarizes technology and system parameters. Parameters N_{min} and N_{max} are set by the circuit breaker's tripping curve. Parameters p_c and p_r are set by the chip's cooling mechanism and the rack's UPS batteries. These probabilities decrease as cooling efficiency and recharge speed increase – see Section 3.2.

3.6 Evaluation

We evaluate the sprinting game and its equilibrium threshold against several alternatives. Although there is little prior work in managing sprints, we compare against three heuristics that represent broader perspectives on power management. First, greedy heuristics focus on the present and neglect the future [51]. Second, control-theoretic heuristics are reactive rather than proactive [78, 79]. Third, centralized heuristics focus on the system and neglect individuals. Unlike these approaches, the sprinting game anticipates the future and emphasizes strategic agents who participate in a shared system.

Greedy (G) permits agents to sprint as long as the chip is not cooling and the rack is not recovering. This mechanism may frequently trip the breaker and require rack recovery. After recovery, agent wake-ups and sprints are staggered across two epochs. Greedy produces a poor equilibrium—knowing that everyone is sprinting, an agent's best response is to sprint as well.

Exponential Backoff (E-B) throttles the frequency at which agents sprint. An agent sprints greedily until the breaker trips. After the first trip, agents wait $0-1$ epoch before sprinting again. After the second trip, agents wait $0-3$ epochs. After the t-th trip, agents wait for some number of epochs drawn randomly from $[0,\, 2^t - 1]$. The waiting interval contracts by half if the breaker has not been tripped in the past 100 epochs.

Cooperative Threshold (C-T) assigns each agent the globally optimal threshold for sprinting. The coordinator exhaustively searches for the threshold that maximizes system performance. The coordinator enforces these thresholds although they do not reflect agents' best responses to system dynamics. These thresholds do not produce an equilibrium but do provide an upper bound on performance.

Equilibrium Threshold (E-T) assigns each agent her optimal threshold from the sprinting game. The coordinator collects performance profiles and implements Algorithm 1 to produce thresholds that reflect agents' best responses to system dynamics. These thresholds produce an equilibrium and agents cannot benefit by deviating from their assigned strategy.

3.6.1 Sprinting Behavior

Figure 3.6 compares sprinting policies and resulting system dynamics as 1000 instances of *Decision Tree*, a representative application, computes for a sequence of

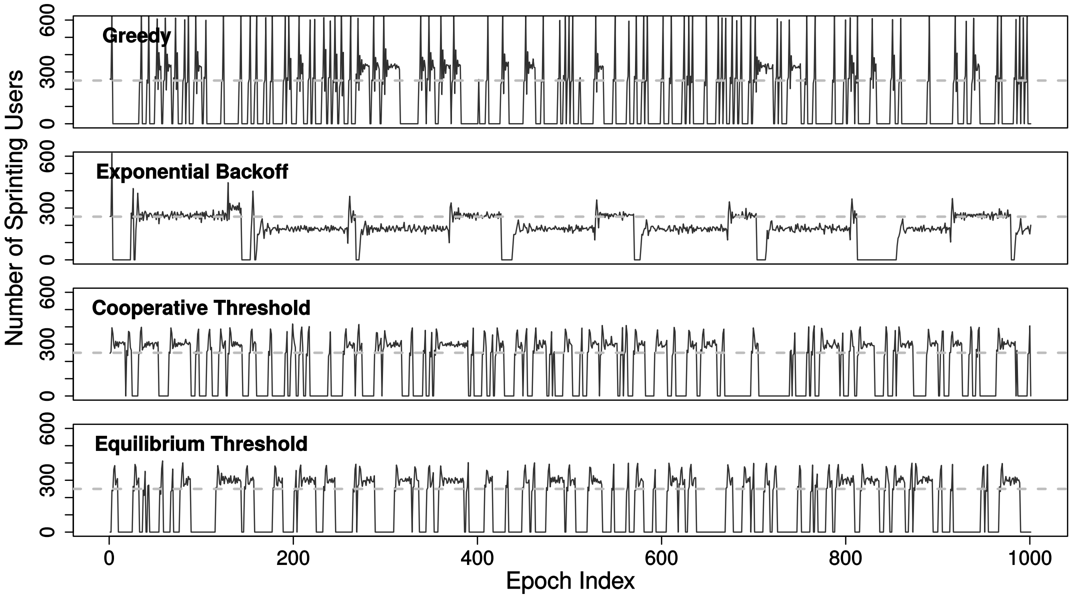


FIGURE 3.6: **Sprinting Behavior.** Shown for a representative application, Decision Tree. Black line denotes number of sprinters. Grey line denotes the point at which sprinters risk a power emergency, $N_{\min}$.

epochs. Sprinting policies determine how often agents sprint and whether sprints trigger emergencies. Ideally, policies would permit agents to sprint up until they trip the circuit breaker. In this example, 250 of the 1000 agents for *Decision Tree* can sprint before triggering a power emergency.

Greedy heuristics are aggressive and inefficient. A sprint in the present precludes a sprint in the near future, harming subsequent tasks that could have benefited more from the sprint. Moreover, frequent sprints risk power emergencies and require rack-level recovery. G produces an unstable system, oscillating between full-system sprints that trigger emergencies and idle recovery that harms performance. G staggers the distribution of sprinting permissions after recovery to avoids dI/dt problems, which reduces but does not eliminate instability.

Control-theoretic approaches are more conservative, throttling sprints in response to power emergencies. E-B adaptively responds to feedback, producing a more stable system with fewer sprints and emergencies. Indeed, E-B may be too conservative, throttling sprints beyond what is necessary to avoid tripping the circuit breaker. The number of sprinters is consistently lower than N_{min}, which is safe but leaves sprinting opportunities unexploited. Thus, in neither G nor E-B do agents sprint to full advantage.

In contrast, the computational sprinting game performs well by embracing agents' strategies. E-T produces an equilibrium in which agents play their optimal strategies and converge to a stationary distribution. In equilibrium, the number of sprinters is just slightly above $N_{min} = 250$, the number that causes a breaker to transition from the non-tripped region to the tolerance band. After emergency and recovery, the system quickly returns to equilibrium. Note that E-T's system dynamics are similar to those from the high-performance, cooperative C-T policy.

Figure 3.7 shows the percentage of time an agent spends in active, cooling, and recovery states. The analysis highlights G and E-B's limitations. With G, an agent spends more than 50% of its time in recovery, waiting for batteries to recharge after an emergency. With E-B, an agent spends nearly 40% of its time in active mode but not sprinting.

Agents spend comparable shares of their time sprinting in each policy. However, this observation understates the sprinting game's advantage. G and E-B sprint at every opportunity and ignore transitions into cooling states, which preclude sprints in future epochs. In contrast, E-T and C-T's sprints are more timely as strategic agents sprint only when estimated benefits exceed an optimized threshold. Thus, a sprint in E-T or C-T contributes more to performance than one in G or E-B.

3.6.2 Sprinting Performance

Figure 3.8 shows task throughput under varied policies. The sprinting game outperforms greedy heuristics and is competitive with globally optimized heuristics. Rather than sprinting greedily, E-T uses equilibrium thresholds to select more prof-

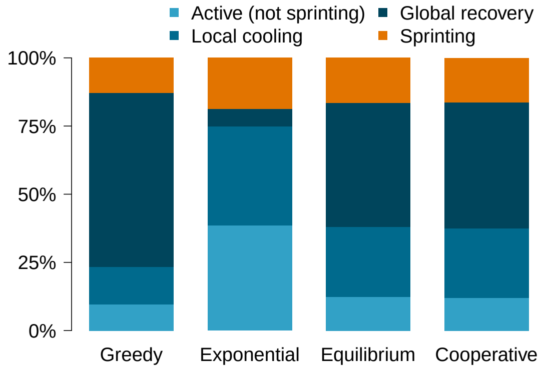


FIGURE 3.7: **Percentage of Time Spent in each State.** Shown for a representative application, Decision Tree.

itable epochs for sprinting. E-T outperforms G and E-B by up to 6.8× and 4.8×, respectively. Agents who use their own strategies to play the game competitively produce outcomes that rival expensive cooperation. E-T's task throughput is 90% that of C-T's for most applications.

Linear Regression and *Correlation* are outliers, achieving only 36% and 65% of cooperative performance. For these applications, E-T performs as badly as G and E-B because the applications' performance profiles exhibit little variance and all epochs benefit similarly from sprinting. When an agent cannot distinguish between epochs, she sets a low threshold and sprints for every epoch. In effect, for such applications, E-T produces a greedy equilibrium.

Thus far, we have considered agents for applications of the same type that compute together. When agents represent different types of applications, E-T assigns different sprinting thresholds for each type. Figure 3.9 shows performance as the number of application types increases. We evaluate performance for a system with k types by randomly selecting k applications, finding each agent's strategy under an E-T policy, and repeating ten times to report an average. As before, E-T performs much better than G and E-B. We do not evaluate C-T because searching for optimal thresholds for multiple types of agents is computationally hard.

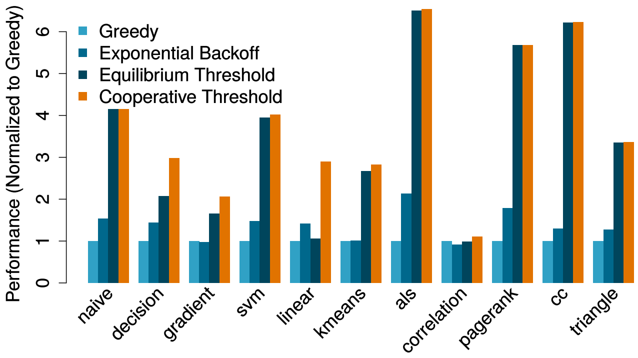


FIGURE 3.8: **Performance.** Measured in tasks per second and normalized against greedy, for a single application type.

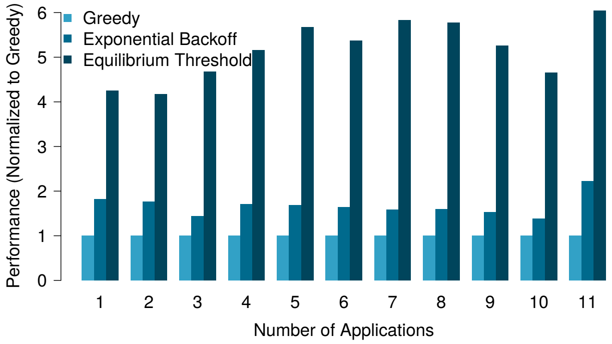


FIGURE 3.9: **Performance.** Measured in tasks per second and normalized against greedy, for multiple application types.

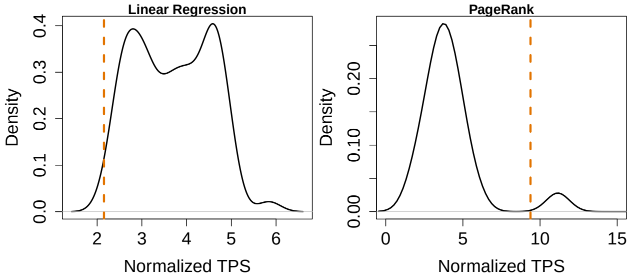


FIGURE 3.10: **Probability Density for Sprinting Speedups.**

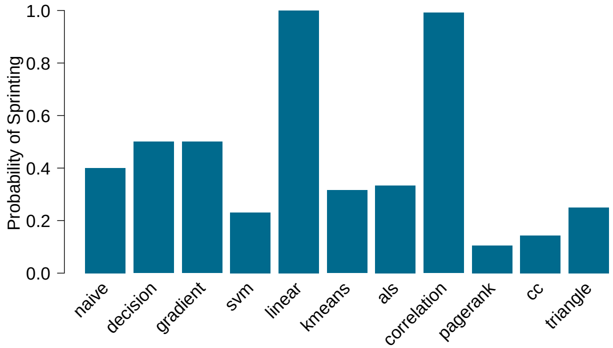


FIGURE 3.11: **Probability of Sprinting.**

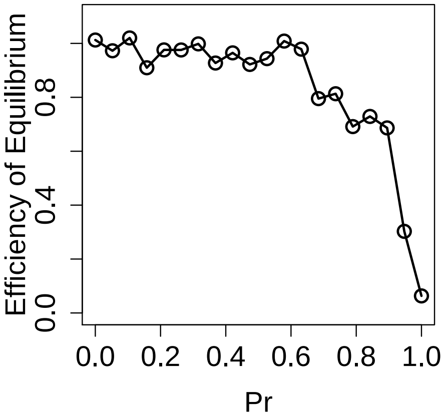


FIGURE 3.12: **Efficiency of Equilibrium Thresholds.**

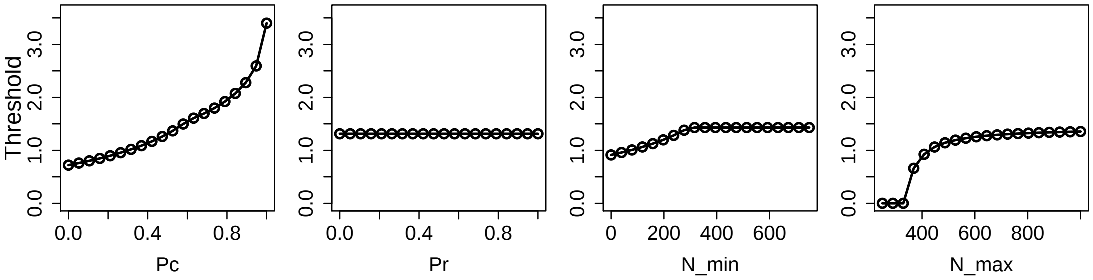


FIGURE 3.13: **Sensitivity of Sprinting Threshold to Architectural Parameters.** Probability of staying in cooling and recovery p_c, p_r and the tripping curve $N_{\min}, N_{\max}$.

3.6.3 Sprinting Strategies

Figure 3.10 uses kernel density plots for two representative applications, *Linear Regression* and *PageRank*, to show how often and how much their tasks benefit from sprinting. *Linear Regression* presents a narrower distribution and performance gains from sprinting vary in a band between 3× and 5×. In contrast, *PageRank*'s performance gains can often exceed 10×.

The coordinator uses performance profiles to optimize threshold strategies. *Linear Regression*'s strategy is aggressive and uses a low threshold that often induces sprints. This strategy arises from its relatively low variance in performance gains. If sprinting's benefits are indistinguishable across tasks and epochs, an agent sprints indiscriminately and at every opportunity. *PageRank*'s strategy is more nuanced and uses a high threshold, which cuts her bimodal distribution and implements judicious sprinting. She sprints for tasks and epochs that benefit most (*i.e.*, those that see performance gains greater than 10×).

Figure 3.11 illustrates diversity in agents' strategies by reporting their propensities to sprint. *Linear Regression* and *Correlation*'s narrow density functions and low thresholds cause these applications to sprint at every opportunity. The majority of applications, however, resemble *PageRank* with higher thresholds and judicious sprints.

3.6.4 Equilibrium versus Cooperation

Sprinting thresholds from equilibria are robust to strategic behavior and perform well. However, cooperative thresholds that optimize system throughput can perform even better. Our evaluation has shown that the sprinting game delivers 90% of the performance from cooperation. But we find that the game performs well only when the penalties from non-cooperative behavior are low. To understand this insight, let us informally define efficiency as the ratio of game performance from equilibrium thresholds (E-T) to optimal performance from cooperative thresholds (C-T).[1]

The sprinting game produces efficient equilibria because the penalty for non-cooperative behavior is triggering a power emergency. In the sprinting architecture, recovery is relatively inexpensive as batteries recharge and normal system operation resumes in ten epochs or less. However, higher penalties for non-cooperative behavior would degrade the game's performance from equilibrium strategies. Figure 3.12 shows how efficiency falls as recovery from power emergencies become increasingly expensive. Recall that p_r is the probability an agent in recovery stays in that state.

Prisoner's Dilemma. The sprinting game fails when an emergency requires indefinite recovery and p_r is one. In this extreme scenario, we would like the game to produce an equilibrium in which agents sprint yet avoid tripping the breaker. Unfortunately, the game has no equilibrium that avoids tripping the breaker and triggering indefinite recovery. If a strategic agent were to observe system dynamics that avoid tripping the breaker, which means P_{trip} is zero, she would realize that other agents have set high thresholds to avoid sprints. Her best response would be lowering her threshold and sprinting more often. Others would behave similarly and drive P_{trip} higher.

In equilibrium, P_{trip} would rise above zero and agents would eventually trip the breaker, putting the system into indefinite recovery. Thus, selfish agents would produce inefficient equilibria—the Prisoner's Dilemma in which each agent's best response performs worse than a cooperative one.

Enforcing Non-Equilibrium Strategies. The Folk theorem guides agents to a more efficient equilibrium by punishing agents whose responses harm the system. The coordinator would assign agents the best cooperative thresholds to maximize system performance from sprinting. When an agent deviates, she is punished such

[1] We are informal because the domain of strategies is huge and we consider only the best cooperative threshold. A non-threshold strategy might provide even better performance.

that performance lost exceeds performance gained. When applied to our previous example, punishments would allow the system to escape inefficient equilibria as agents are compelled to increase their thresholds and ensure P_{trip} remains zero.

Note that threat of punishment is sufficient to shape the equilibrium. Agents would adapt strategies based on the threat to avoid punishment. The coordinator could monitor sprints, detect deviations from assigned strategies, and forbid agents who deviate from ever sprinting again. Alternatively, agents could impose collective punishment by continuously sprinting, triggering emergencies, and degrading everyone's performance. The threat of collective action deters agents who would deviate from the cooperative strategy.

3.6.5 Sensitivity Analysis

Figure 3.13 shows the sprinting threshold's sensitivity to the game's parameters. In practice, server engineering affects cooling and recovery durations (p_c, p_r) as well as the breaker's trip curve ($N_{\min}$, $N_{\max}$).

As cooling duration increases, thresholds increase and agents sprint less. Agents are more cautious because sprinting in the current epoch requires many more epochs for cooling. The opportunity cost of sprinting mistakenly rises. As recovery duration increases, the cost of tripping the breaker increases. However, because each agent sprints to pursue her own performance while hoping others do not trip the breaker, thresholds are insensitive to recovery cost. When p_r is one, we have shown how agents encounter the Prisoner's Dilemma – see Section 3.6.4.

When $N_{\min}$ and $N_{\max}$ are small, the probability of tripping the breaker is high. Ironically, agents sprint more aggressively and extract performance now because emergencies that forbid future sprints are likely. When $N_{\min}$ and $N_{\max}$ are big, each agent sprints more judiciously as a sprint now affects the ability to sprint in the future.

3.7 Related Work

The sprinting problem falls into the general category of datacenter power management, but we are the first to identify the problem and propose a game-theoretic approach. The sprinting problem is made interesting by modern approaches to datacenter provisioning.

To minimize total cost of ownership and maximize return on investment, datacenters oversubscribe their servers [12, 80], bandwidth [81], branch circuits [60], cooling and power supplies [48, 64]. In datacenters, dynamic power capping [82] adjusts the power allocation to individual servers, enabling a rich policy space for power and energy management. In servers, managers could pursue performance while minimizing operating costs, which are incurred from energy and cooling [83, 84, 85, 86]. Researchers have sought to allocate server power to performance critical services via DVFS [59, 87].

Economics and game theory have proven effective in datacenter power and resource management [46]. Price theory [88] have been applied to manage heterogeneous server cores. Demand response models have been proposed to handle power emergencies [89]. In addition to performance, fairness in game theory has been studied to incentivize users when sharing hardware in a cloud environment [8, 10, 1].

In this chapter, we treat the sprinting management problem as a repeated game and seek an equilibrium that leads sprinting servers to expected behavior. Similar approaches have been applied to power control in wireless communication systems [90]. But we are the first to consider game theory for datacenter management, especially in the context of computational sprinting and power capping.

3.8 Conclusions

We present a sprinting architecture in which many, independent chip multiprocessors share a power supply. When an individual chip sprints, its excess heat constrains future sprints. When a collection of chips sprint, its additional power demands raise the risk of power emergencies. For such an architecture, we present a management framework that determines when each chip should sprint.

We formalize sprint management as a repeated game. Agents represent chip multiprocessors and their workloads, executing sprints strategically on their behalf. Strategic behaviors produce an equilibrium in the game. We show that, in equilibrium, the computational sprinting game outperforms prior, greedy mechanisms by 4-6× and delivers 90% of the performance achieved from a more expensive, globally enforced mechanism.

4

Managing Heterogeneous Datacenters with Tokens

4.1 Introduction

Processor heterogeneity is a fundamental design strategy at all scales, from chip multiprocessors to warehouse-scale datacenters. Heterogeneity improves performance and energy efficiency when tasks compute in their best suited setting. But managing heterogeneity is challenging. Prior efforts pursue energy efficiency [91, 92], instruction throughput in chip multiprocessors [17], or service quality in datacenters [93, 94].

The pursuit of fairness in heterogeneous systems is equally important but less understood. Fairness encourages users to dynamically share systems. If a user dislikes an allocation policy, she may prefer private or statically partitioned systems, which are less efficient than dynamically shared ones. Users seek at least two assurances—sharing incentives (SI) and envy-freeness (EF) [15]. SI ensure that users perform at least as well as they would have under equal division. EF ensures that users prefer their own allocation over other users'.

Recently, researchers have turned to microeconomics and game theory for fair, multi-resource allocation [8, 1, 21, 20, 41]. These studies pursue fairness in space, dividing hardware resources that outnumber software tasks. In contrast, we pursue fairness in time for heterogeneous systems that multiplex scarce hardware to satisfy simultaneous, competing demands. For example, consider a system where the majority of processors are "small" and only a few are "big."

Fairness versus Performance. Round-robin is the classic policy for managing scarce resources over time. It divides time into rounds and assigns each user their fair share of rounds in fixed rotation. Although round-robin guarantees users equal time on preferred processors, as we show in this chapter, it fails to ensure EF. Moreover, round-robin performs poorly in heterogeneous systems because it ignores phases in users' workloads. Some workloads receive big processors when small ones would have sufficed.

Randomized and proportional shares, with a mechanism like lottery scheduling [32], ensure that users receive equal shares over time. In a system with n users and m big processors, $m < n$, each user receives a big processor with probability m/n in each round. Ex ante, allocations guarantee SI and EF in expectation over multiple rounds. However, performance suffers in much the same way it does with round-robin.

Equal-progress is another classic policy that balances performance across parallel tasks, which helps manage a job's critical path. However, it fails to produce the requisite conditions for sharing in multi-user, multi-program settings. Specifically, equal-progress unfairly favors tasks that require large allocations for progress, which provides neither SI nor EF [18, 1].

Allocation Games. Performance losses from fair allocation are inevitable when there is no information about workloads' utilities for big processors. But what if users know their workloads' utility distributions from resources over time? What if users know their workloads' utilities for a resource in the current round? Our answer is a repeated game that extracts and uses information about workloads' utilities to simultaneously improve performance and ensure fairness.

We frame heterogeneous processor allocation as a repeated game and study users' strategies for requesting processors. In equilibrium, the game improves performance by increasing system flexibility. It allocates resources to users that benefit most in each round and compensates those that are unfairly treated with more resources in later rounds. For a case study, the game manages processors with heterogeneous power budgets—"small" processors operate within a modest budget and "big" processors operate within an augmented one. The following lists our contributions.

- **Repeated Game for Managing Heterogeneity.** We propose a repeated game in which users spend tokens when allocated big processors. We apply the game to allocate power boosts. (Sections 4.2–4.3)

- **Tokens and Game Theory.** We formulate the repeated game for heterogeneous processor allocation. We optimize users' strategies for spending tokens and identify conditions for equilibria. (Section 4.4)

- **Tokens in Practice.** We describe a framework that profiles tasks, optimizes strategies, and allocates power. Profiling and optimization are offline whereas allocation is online. (Section 4.5)

- **Fairness and Performance.** The allocation game incentivizes sharing and mitigates ex post envy over time. The game performs much better than other fair policies, (e.g., 1.7x better than round-robin) and performs comparably to performance-maximizing policies. (Sections 4.6–4.7)

Collectively, the results suggest potential for economic game theory in systems resource management. The game extends naturally to any setting in which applications compete for scarce, preferred hardware.

4.2 Motivation and Background

System Setting. We study systems with many small processors and a few big ones. Each user can receive a processor, just not necessarily a big one. Big and small processors arise in many settings, from chip multiprocessors to warehouse scale datacenters. Our mechanism is particularly well suited to managing processors that dynamically re-configure between big and small capability (e.g., with dynamic voltage/frequency scaling).

Allocation Games. We formulate the allocation of heterogeneous processors as a repeated game between strategic users. Rather than burden human users with a complex space of actions, we design agents to represent users and their jobs in the shared system. Each agent selfishly requests processors to maximize individual performance subject to the rules of the game.

Agents analyze workload phases, which cause their utility from heterogeneous processors to vary over time. Ideally, agents request big processors when they are most beneficial and, all else being equal, prefer a big processor in the present over one in the future. Responding to agents' requests in each round, the game (re-)allocates processors. Allocations in the present affect those in future. In turn, agents adapt their strategies for requesting resources according to competitive dynamics. Over time, strategic game play can produce an equilibrium.

Game-Theoretic Desiderata. We focus on fair resource allocations that encourage participation in shared systems. In microeconomic theory [15], fairness is defined by sharing incentives (SI), when every agent's allocation performs at least as well as it would have under equal division, and envy-freeness (EF), when every agent prefers its own allocation over another's. Without these axiomatic properties, strategic users may prefer private or statically partitioned systems.

Finding allocations that incentivize sharing in practical systems is challenging, and two recent studies are representative. Ghodsi et al. propose Dominant Resource Fairness when allocating processor cores and memory capacity in distributed systems. [8]. Zahedi et al. propose Resource Elasticity Fairness when allocating cache capacity and memory bandwidth in chip multiprocessors [1]. These algorithms guarantee SI and EF for very different performance models, pursuing fairness in space.

Repeated Games. In this chapter, we pursue fairness in time. We design a game that encourages sharing and mitigates envy across repeated allocations of heterogeneous processors. Because each agent's allocation is a sequence of big and small processors, we add a temporal dimension to SI and EF. The allocation sequence provides repeated sharing incentives (SI_{R}) when every agent performs at least as well as it would have under equal division of time on big processors. And it provides repeated envy-freeness (EF_{R}) when every agent prefers its allocation sequence over another's. Repeated games that flexibly pursue these properties in expectation (i.e., averaged) over rounds are fair and perform better than mechanisms, such as lottery scheduling [32], that rigidly enforce SI and EF in every round.

4.3 Repeated Game with Tokens

The repeated game is implemented with a token mechanism. Tokens determine the allocation of *boosts*, the acceleration from a big processor over a small one. Boosts can be defined by heterogeneity across design generations[95, 96, 93], adaptive microarchitectures [97, 98, 99], and power budgets [2, 49, 100].

Without loss of generality, we describe our token mechanism with a case study in power. Consider a datacenter power delivery unit (PDU) that supplies limited power to m chip multiprocessors. Each multiprocessor runs with nominal power. Additionally, the supply can support n simultaneous power boosts ($n << m$).

Each user runs her job on a chip multiprocessor. Agents represent users and their jobs. Agents' utilities from boosts vary across job phases and rounds. Time is divided into rounds. We assume each agent knows its utility distribution over time and its utility at the beginning of each round, but the management mechanism lacks this information.

4.3.1 Token Mechanism

In each round, agents signal preferences for boosts. The game uses agents' signals and token holdings to allocate boosts. Agents must spend tokens when allocated boosts and may receive tokens otherwise.

Signals (Y and ¬ Y). When beginning a round, each agent signals yes (Y) or no ($\neg\, Y$) to indicate her preference for one of n boosts. If fewer than n agents signal Y, each agent who prefers a boost receives one. If more than n agents signal Y, the n agents holding the most tokens receive boosts.

Tokens (t). Each agent holds a number of tokens t. Agents start with an equal number of tokens.[1] An agent must hold at least one token to signal Y and must spend one token when allocated a boost. The game prohibits hoarding and allocates boosts to agents with $t_{\max}$ tokens, requiring them to spend regardless of their preference.

Token Distribution ($\mathbf{f(t)}$). Agents' token holdings determine winners in the competition for boosts. An agent with few tokens can signal Y but fail to receive a boost when others with more tokens also signal Y. Let $f(t)$ describe the token distribution, which quantifies the percentage of agents with t tokens. An agent with t' tokens is outranked by $\sum_{t>t'} f(t)$ percent of the game's agents, who receive power boosts with higher priority when signaling Y.

Token Redistribution ($\mathbf{P_R}$). In each round, the game redistributes spent tokens among agents who do not receive a boost. When the game allocates n boosts, it redistributes n tokens to the $m-n$ agents who did not receive them with probability

[1] Users could start with different token holdings if they have different weights (*i.e.*, priorities)—see Section 4.9.

$P_R = n/(m-n)$. The game does not use fractional tokens and redistributes tokens probabilistically, which is fair in expectation.

4.3.2 Threshold Strategies

Agents play the game with threshold strategies. A user signals for boosts when its utility exceeds some threshold. Such strategies are trivial to implement at run-time. But determining the optimal threshold requires an offline analysis of the system's competitive dynamics. When beginning a round, each agent's decision to signal depends on the game's history and competitive factors, including the

- agent's tokens,
- agent's utilities from boosts,
- other agents' tokens,
- other agents' utilities and strategies.

An agent's token holdings affect her signaling strategy. An agent who holds many tokens feels rich and signals Y even when utility from a boost is modest. An agent who holds few tokens feels poor, hoards tokens, and signals $\neg\, Y$ even when it would benefit from a boost. An agent signals strategically because it must spend a token if it receives a boost, which affects both its performance in the current round and its ability to signal effectively in future rounds.

Agents signal and exchange tokens, actions which determine how efficiently the game allocates boosts. Suppose agents hoard tokens and rarely request boosts, or spend tokens liberally and often request boosts. Because naive signals do not reveal agents' relative utilities, they produce an uninformative token distribution and the game can do no better than round-robin allocation. Agents who signal strategically, in contrast, request boosts only for rounds that benefit.

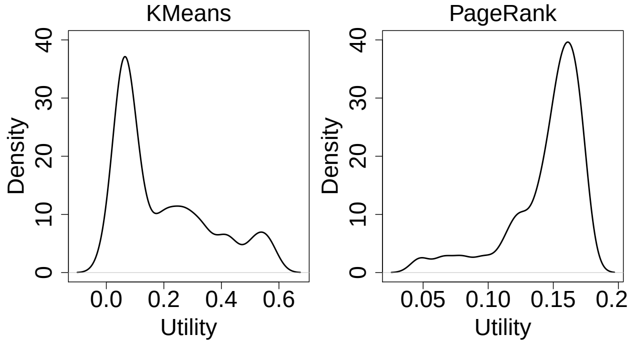


FIGURE 4.1: **Probability Density Functions on Utility.** For example, throughput gains from power boosts.

4.3.3 Game Dynamics

We optimize each agent's strategy in Section 4.4 but first present an extended case study that illustrates game play. See Section 4.6 for more details on experimental methods.

Suppose that the nominal power budget is 30W and a boost is an additional 50W of power. In other words, a little processor has a 30W budget and a big processor has an 80W budget. A 35KW datacenter rack supports 900 little processors and 100 big ones. In each round, 1000 agents compete for 100 power boosts. These parameters reflect modern power supplies [48, 51] and our workloads' typical power demands.

Workloads' Preferences. For insight into competing demands for power, consider two representative Spark workloads and their utilities from boosts. Figure 4.1 shows each agent's probability density on utility u, measuring gains in normalized task throughput from power boosts. `KMeans` agents strongly prefer power boosts whereas `PageRank` agents weakly prefer them. Specifically, `KMeans`'s u ranges from 0.1 to 0.5 whereas `PageRank`'s ranges from 0 to 0.2.

Signaling Strategies. Suppose 500 agents for `KMeans` and 500 agents for `PageRank` play the game by signaling for power boosts when their expected utilities exceed some threshold. Figure 4.2 presents representative thresholds, which

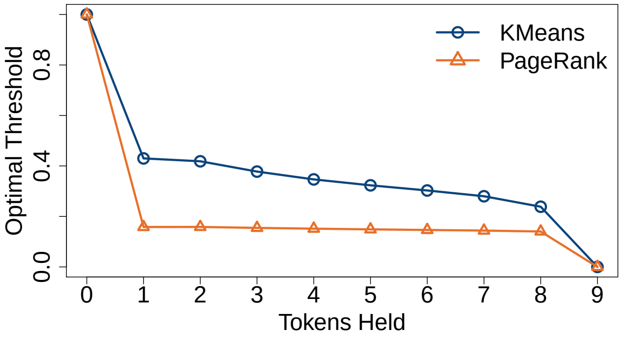


FIGURE 4.2: **Signaling Thresholds.** Agents with more tokens lower thresholds, spend tokens more freely.

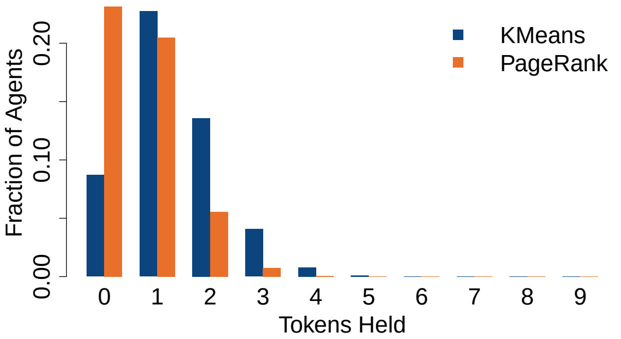


FIGURE 4.3: **Tokens.** `KMeans` agents have higher thresholds, signal less for boosts, receive them less frequently, hold more tokens.

tend to decrease with token holdings. Poor agents signal for boosts only when benefits are large whereas rich agents signal more freely. An agent without tokens cannot receive boosts and a high threshold prevents it from signaling. Agents with t_{max} tokens must receive a boost. A low threshold ensures they signal.

Because the supply of power boosts is limited, `KMeans` agents use higher thresholds to decide when to receive their fair share. With higher thresholds, agents signal for boosts less often and signal when benefits are greater. With lower thresholds, users would spend tokens quickly and exhaust their share of boosts.

Token Distributions. Figure 4.3 illustrates the distribution of tokens after multiple rounds of game play. Agents begin the game with one token and signal for boosts according to their threshold strategies. `KMeans` agents conserve their tokens

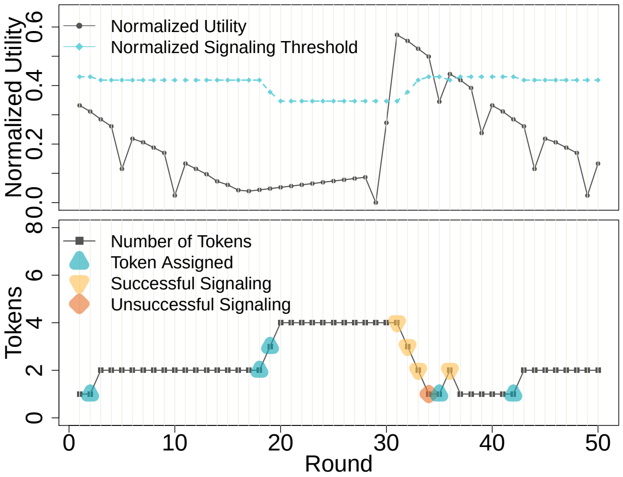


FIGURE 4.4: **Snapshot of System Dynamics.**

and do not signal in every round. PageRank agents have fewer tokens because they signal whenever their utilities are higher than their relatively low thresholds. Among 500 PageRank agents, approximately 35% have zero tokens and 20% have one.

Game Play. Figure 4.4 presents a snapshot of game play for a KMeans agent. The top sub-figure superimposes utility and threshold, which vary over time. The bottom sub-figure shows token holdings and signals for boosts. The agent signals when utility exceeds threshold. It does not signal in rounds 0 to 30, accumulating tokens to ensure successful signals when boosts are most needed in rounds 30 to 35, an exceptionally compute-intensive phase.

Token holdings and signaling thresholds fluctuate during game play. Holdings increase when an agent's utility is low and it does not signal while others do (e.g., rounds 0 to 30). Tokens spent by other agents are redistributed. Holdings decrease when an agent's utility is high and it signals for boosts (e.g., rounds 30 to 35). Note that thresholds rise as token holdings fall because the agent signals and spends tokens more judiciously.

Not all signals are successful. An agent may signal yet fail to receive a boost when others hold more tokens. We observe a series of successful signals beginning in round 31 followed by an unsuccessful signal in round 34. Prior successes reduce token holdings such that the agent is outranked by others who signal.

4.4 Strategic Game Play

We optimize agents' threshold strategies and find the game's equilibrium. First, we derive equations that describe game play. Then, we use these equations to refine thresholds and assess outcomes. Finally, we produce the optimal threshold for each agent. These thresholds produce an equilibrium in which no agent benefits by deviating from its assigned strategy.

Although the analysis is sophisticated, it has low overheads. The models do not delay resource allocation because thresholds are optimized offline and only checked online. The offline computation requires a few seconds, using a dynamic program to refine thresholds.

The analysis is efficient despite the difficulty of the problem. Each agent must identify its best action from a complex strategy space. And it must respond to competitors' actions but cannot tractably monitor every other agent (e.g., token holdings, signals) in large systems with many participants.

Equilibrium. Addressing these challenges, we study the mean field equilibrium (MFE), a solution built atop statistical summaries of the game [67, 68, 69, 70, 71, 72]. Each agent optimizes strategies against expectations of population behavior. We find a MFE by analyzing interdependent distributions that describe agents.

- **Signaling Strategy ($\mathbf{P_Y(t)}$).** Probability agent signals Y for boost when holding t tokens.

- **Token Distribution ($\mathbf{f(t)}$).** Statistical distribution measuring fraction of agents with t tokens.

- **Signaling Strength ($\mathbf{P_B(t)}$).** Probability agent holding t tokens receives boost when signaling Y.

In equilibrium, agents are described by stationary distributions, which are invariant across time. For example, some agents spend tokens and others receive them in a given round, but the token distribution is unchanged with game play across rounds.

We find an equilibrium by finding the game's stationary distributions. Specifically, we construct an algorithm that characterizes agents and optimizes each agent's response to the population as follows.

- **Optimize Signaling Strategy ($\mathbf{P_B \rightarrow P_Y}$).** Given signaling strength, determine strategy to maximize utility.

- **Assess Token Distribution ($\mathbf{P_Y \rightarrow f}$).** Given signaling strategy, spend tokens and determine token distribution.

- **Assess Signaling Strength ($\mathbf{P_Y, f \rightarrow P'_B}$).** Given signaling strategy and token distribution, determine signaling strength.

- **Iterate ($\mathbf{P'_B \rightarrow P_B}$).** If $P_B = P'_B$, distributions are stationary and game is in equilibrium. Otherwise, iterate with new P_B.

The algorithm iteratively refines an agent's strategy. In response to its expected signaling strength, each agent optimizes its signaling strategy to maximize performance. Signals affect token holdings, which in turn affect signaling strength. The algorithm terminates with an equilibrium when the game converges to stationary distributions and agents find their optimal strategies.

We detail each step of the algorithm in the following sections. *Utility* refers to performance gain from power boost. *Value* is an agent's expected utility in the present based on statistical estimates of the future.

4.4.1 Optimize Signaling Strategy

For each agent, the Bellman equation determines whether signaling for a power boost maximizes value V given its token holdings t and boosted utility u. V_Y and $V_{\neg Y}$ are values when signaling Y and $\neg Y$.

$$V(t,\ u) = \max(V_Y(t,\ u), V_{\neg Y}(t,\ u)) \tag{4.1}$$

If $V_Y(t,\ u) \geq V_{\neg Y}(t,\ u)$, then signaling Y is optimal in state $(t,\ u)$. Otherwise, signaling $\neg Y$ is optimal. We determine the optimal signal for every $(t,\ u)$ with dynamic programming. The resulting map from state to signal specifies the optimal strategy.

Value from Signal. The value from signaling Y depends on whether the signal is successful. The signal succeeds with probability $P_B(t)$, producing utility u from boosted performance in the current round plus future value after spending one token $\gamma\ V(t-1)$. Future values are discounted by $\gamma < 1$ because agents prefer utility now over utility later, all else being equal.

$$\begin{aligned} V_Y(t,\ u) \ &= \ P_B(t)\Big(u + \gamma\ V(t-1)\Big) \ + \\ &(1 - P_B(t))\Big(P_R\ \gamma\ V(t+1) + (1-P_R)\ \gamma\ V(t)\Big) \end{aligned} \tag{4.2}$$

The signal fails with probability $1 - P_B(t)$, producing no utility. After failure, future value depends on how tokens are redistributed; the agent receives a token with probability P_R.

$V(t)$ is the expected future value when holding t tokens. Maximizing value is complicated by uncertainty and incomplete task profiles. Future value cannot be known precisely because utility u varies across computational phases. However, profilers that measure performance across rounds can supply distribution $h(u)$, the probability that boosted performance is u.

$$V(t) = \mathbb{E}\left[V(t,\ u)\right] = \int V(t,\ u)\ h(u)\ du$$

We similarly assess value from not signaling. An agent who signals $\neg Y$ derives the same value as an agent who signals Y but fails to receive a boost.

$$V_{\neg Y}(t,\ u) = \gamma\ \big(P_R\ V(t+1) + (1-P_R)\ V(t)\big) \tag{4.3}$$

Threshold Strategy. Dynamic programming produces the optimal signaling strategy – a threshold on performance gains from power boosts. Specifically, agents maximize value by signaling for boosts when $V_Y > V_{\neg Y}$. From Equations (4.2)–(4.3), agents should signal if boosted utility u exceeds threshold u_{thr}.

$$u\ \ >\ \ \underbrace{\gamma\ \big(P_R V(t+1) + (1-P_R)V(t) - V(t-1)\big)}_{u_{\text{thr}}(t)} \tag{4.4}$$

Note that the threshold varies with the agent's token holdings t and utility distribution $h(u)$.

$$P_Y(t) = \Pr\big(u \geq u_{\text{thr}}(t)\big) = \int_{u \geq u_{\text{thr}}(t)} h(u)\ du. \tag{4.5}$$

Given its threshold, an agent signals for boosts with probability $P_Y(t)$. This probability depends on several factors. First, threshold u_{thr} specifies the gains required to justify signals. Second, token holding t affects the threshold as rich agents set lower thresholds to signal more liberally. Finally, utility distribution $h(u)$ affects the threshold. Agents that often benefit from boosts set higher thresholds, judiciously signaling in rounds that benefit most.

Agents implement their signaling strategies with offline analysis and online comparisons, sketched here and detailed in Section 4.5. Offline, agents profile boosted performance, construct utility distributions, and optimize thresholds. This computation requires a few seconds. Online, in each round, the agent signals when utility exceeds threshold, a comparison that requires modest support from hardware counters.

4.4.2 Assess Token Distribution

Agents exchange tokens as the game allocates power boosts. Figure 4.5 presents a Markov chain that specifies possible token holdings and transitions between them.

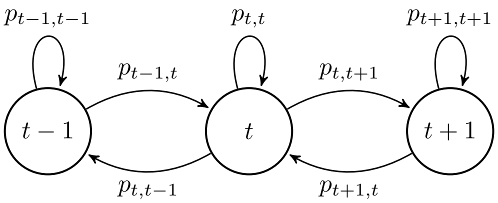


FIGURE 4.5: **Token Exchange.**

An agent with i tokens has j tokens in the next round with probability $p_{i,j}$.

$$\begin{aligned} p_{t,t-1} &= P_Y(t)\ P_B(t) \\ p_{t,t} &= (1-P_R)\ \big(1-P_Y(t)\ P_B(t)\big) \\ p_{t,t+1} &= P_R\ \big(1-P_Y(t)P_B(t)\big) \end{aligned} \tag{4.6}$$

An agent loses a token after signaling successfully. Otherwise, it gains a token with some probability. The net change in token holdings depends on probability of signaling P_Y, receiving a boost P_B, or a token P_R.

When agents signal using optimized thresholds, the Markov chain converges to a stationary distribution $f(t)$, characterized by linear equations for $t \in (0,\ t_{\max})$.

$$f(t) = f(t+1)\ p_{t+1,t} + f(t)\ p_{t,t} + f(t-1)\ p_{t-1,t} \tag{4.7}$$

We omit boundary conditions for $t = 0$ and $t = t_{\max}$, which differ only slightly. Agents hold a non-negative number of tokens and hold no more than $t_{\max}$ tokens. An agent with $t_{\max}$ tokens receives a boost and spends a token.

4.4.3 Assess Signaling Strength

Signaling strength is the probability of successfully requesting a power boost. We calculate strength from its signaling strategy and token holdings. First, we define the probability an agent holds t tokens and signals Y.

$$g(t) = f(t)\ P_Y(t)$$

Then, we determine the percentage of agents who signal Y and hold at least t tokens.

$$G(t) = \sum_{t'=t}^{t_{\max}} g(t')$$

Algorithm 2: Optimize Signaling Strategy

```
input  : Initial tokens per agent $t_{ini}$;
         Utility distribution per agent $h(u)$;
         Token redistribution $P_R = n/(m-n)$
output: Optimal threshold $u_{\text{thr}}$
while $P_B$ not converged do
    $u_{\text{thr}} \leftarrow$ DP for Equations (4.1)–(4.4) given $P_B$
    $P_Y \leftarrow$ Equation (4.5) given $h(u)$, $u_{\text{thr}}$
    $f(t_{ini}) \leftarrow 1$
    while $f$ not converged do
        $f \leftarrow$ Equations (4.6)–(4.7) given $P_Y$, $P_B$, $f$
        $P_B \leftarrow$ Equation (4.8) given $f$, $P_Y$
    end
end
```

Finally, we determine the probability an agent signals successfully for one of m boosts when holding t tokens.

$$P_B(t) = \begin{cases} 0 & \text{if } mG(t+1) \geq n, \\ 1 & \text{if } mG(t) < n, \\ \frac{n-mG(t+1)}{mg(t)} & \text{otherwise.} \end{cases} \tag{4.8}$$

The definition of $P_B(t)$ enumerates scenarios for an agent with t tokens. First, the agent fails to receive a boost when agents that signal with $> t$ tokens outnumber available boosts. Second, if agents that signal with $\geq t$ tokens undersubscribe boosts, all of these agents receive them. Finally, ties are broken for agents with t tokens. The $mG(t+1)$ signaling agents with $> t$ tokens receive boosts. The remaining $n - mG(t+1)$ boosts are assigned with equal probability to the $mg(t)$ agents with t tokens.

4.4.4 Equilibrium

The mean field equilibrium is defined by stationary distributions for signaling strategy, token holdings, and signaling strength. The game is in equilibrium if

- P_Y is signaling strategy that uses threshold u_{thr} to solve Equations (4.1)–(4.3).

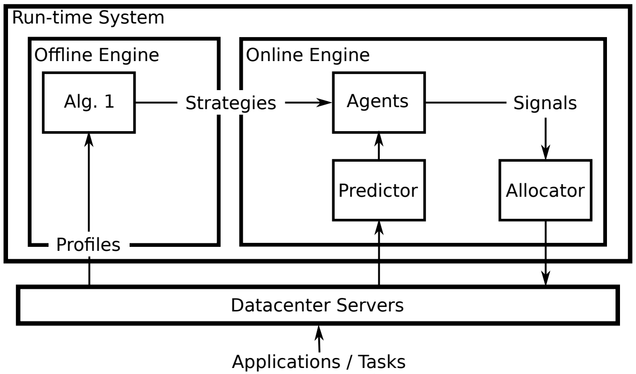


FIGURE 4.6: **Game Architecture.**

- f is token distribution that satisfies consistency conditions in Equations (4.6)–(4.7).
- P_B is signaling strength that satisfies consistency conditions in Equation (4.8).

Algorithm 2 optimizes signaling strategies to produce an equilibrium. Given an initial P_B, the outer loop optimizes threshold u_{thr} and corresponding strategy P_Y. Given this strategy, the inner loop finds stationary token distribution f and assesses signaling strength P_B. The algorithm iteratively updates P_Y, f, and P_B until they converge to stationary distributions and satisfy equilibrium conditions.

To prove that the mean field equilibrium exists and Algorithm 2 finds the equilibrium, we need to use a fixed point theorem. This is out of the scope of this chapter and is a future work in the field of theoretical game theory. In practice, the algorithm converges within tens of loop iterations for every workload we study. We cap the number of loop iterations (e.g., $I_{\max}$) and, if the algorithm fails to converge, we default to randomly signal for boosts, implementing probabilistic round-robin.

4.5 Game Architecture

Figure 4.6 summarizes the token system's architecture in practice. The offline engine profiles workload utilities and optimizes agents' signaling strategies by identifying their best response to other agents. The online engine compares expected utility against thresholds that define agents' optimized strategies. It then allocates boosts according to signals and relative token holdings. The allocator enforces management decisions with power capping.

4.5.1 Offline: Optimizing Strategies

The offline engine profiles each agent's workload and utility over time, producing a probability density for utility from power boost ($h(u)$). Our baseline profiler runs the computation twice, once with the power boost and again under nominal power, and compares performance counters in every round.

In future, we could enhance profilers to learn $h(u)$ as computation progresses. Agents could begin game play by reporting constant utility, which indicates no information about $h(u)$. As the game proceeds, agents would observe boosted performance and update the probability density.

Implementation. The offline engine runs Algorithm 2 to optimize threshold strategies based on utility distributions. We implement the algorithm in R. Algorithm 2 completes within 10 seconds on an Intel® Core™i7-3630QM 2.4GHz processor. The offline algorithm updates agents with newly optimized strategies when they become available. The algorithm does not affect the critical path in online allocation.

To solve the dynamic program (DP), we use value-iteration with computational complexity that is linear with the number of states and actions when state transition probabilities are sparse [101]. The convergence rate of the method slows as the discount factor γ approaches 1. In the worst case, the number of iterations grows polynomially in $(1-\gamma)^{-1}$.

4.5.2 Online: Signaling and Allocation

The online engine receives and deploys optimized threshold strategies. In each round, it predicts agents' utilities from boosts. Our experiments assume oracular prediction, reading utilities from traces of computation on instrumented servers. In practice, agents could profile their workload under nominal and boosted power budgets for a small portion of each round to predict utility in that round.

Implementation. An agent with t tokens signals Y when its profiled or predicted utility exceeds its signaling threshold, $u > u_{\text{thr}}(t)$. Profiling and predicting utility require hardware counters but is computationally inexpensive. Comparing utility against pre-computed thresholds is trivial.

The game ranks agents by token holdings and allocates boosts to those who signal and hold the most tokens. Sorting m agents requires $\mathrm{O}(m \log m)$ time. A simple approach sorts agents in every round, which takes less than 1ms for 1000 agents and less than 4ms for 10000 agents.

The allocator implements power boosts with Intel RAPL [102]. Writing to the registers requires milliseconds and changing the voltage/frequency requires microseconds. These latencies are negligible compared to rounds that span tens of seconds.

4.6 Experimental Methodology

Spark Workloads. We evaluate task-parallel datacenter workloads from Apache Spark [53]—see Table 4.1. Each user runs a Spark job on a chip multiprocessor, which is managed by an agent who signals to request power boosts on behalf of the user and her workload.

We define performance in terms of relative progress. Specifically, for each round in the game, a job's performance is the number of completed tasks divided by the total number of tasks in the job. This measure places performance on the same scale across Spark jobs, which have tasks that vary in number and size.

We define utility as the gap between utilities under nominal and boosted power. We trace task throughput under these power budgets in each round for each workload. Since the length of the trace is shorter under boosted power, we extend the shorter trace with linear interpolation.

Physical Server Measurements. We run Spark applications on physical machines to profile performance and trace phase behavior. Each server has two sockets, and each socket has an Intel® Xeon®E5-2697 (v2) Processor. We enable power boosts using RAPL [102]. We set power limits for sockets by writing to the `MSR_PKG_POWER_LIMIT` register. We trace utility from power boosts by measuring

Table 4.1: Spark Workloads

ID	Applications	Dataset	Data Size
1	`Correlation`	kdda2010 [73]	2.5G
2	`FP Growth`	Webdocs [103]	1.5G
3	`KMeans`	uscensus1990 [74]	327M
4	`LinearRegression`	kddb2010 [73]	4.8G
5	`ALS`	movielens2015 [75]	325M
6	`SVM`	kdda2010	2.5G
7	`Pagerank`	wdc2012 [76]	5.3G
8	`ConnectedComponents`	wdc2012	5.3G
9	`TriangleCounting`	wdc2012	5.3G

Table 4.2: Experimental Parameters

Description	Symbol	Value
# Agents	m	1000
# Rounds	t	3000
# Power Boosts	n	100
# Initial Tokens per Agent	iniT	1
Discount Factor Rate	γ	0.99
Precision	ϵ	0.01
Maximum # of iterations	$I_{\max}$	200

each workload's task throughput under nominal and boosted power limits.

Datacenter Simulation Methods. We use traces from physical machines to simulate allocation at datacenter scale. Table 4.2 summarizes the simulation of 1000 agents that run varied workloads for 3000 rounds. In each 60-second round, agents compete for 100 boosts.

Simulations assume agents enter the system at varied times and restart computation when jobs complete. The trace-based approach reads utility from system profiles at the beginning of each round. In effect, we assume an oracle that predicts utility from a power boost. This optimistic assumption benefits not only our allocation mechanism but also alternatives that we compare against.

We simulate the case study first described in Section 4.3. The datacenter's power delivery unit (PDU) supplies 35KW of power. Power is shared by 1000 chip multiprocessors, each running with a 30W nominal power budget or an additional 50W boost. The power supply can support only 100 simultaneous boosts.

Workload Mixes and Metrics. We construct agent populations with diverse workloads. Simulating k types of agents means sampling k workloads, uniformly at random, from the broader suite in Table 4.1 and launching an equal number of instances for each. We assess diversity across agent populations by averaging results across 25 workload mixes, constructed with IID samples from the benchmark suite. We report figures of merit averaged across workloads in the mix.

4.7 Evaluation

We evaluate the repeated game and token system, referred to as the **mean field (M-F)** mechanism, when workloads share a power budget and signal for power boosts. We compare against baselines that use alternative definitions for fairness. We also compare against a baseline that optimizes throughput without regard for fairness. The following details these alternatives:

- **Round-Robin (R-R).** Allocate equal number of boosted rounds to each agent in fixed rotation. Agents may receive boosts when less power would have sufficed or fail to receive them when needed.
- **Equal-Progress (E-P).** Allocate as many boosted rounds as needed for equal progress across agents. Progress is measured by normalized throughput. In effect, E-P maximizes minimum progress and ensures max-min fairness. Agents may lack incentives and exhibit envy.
- **Equal-Division (E-D).** Divide power equally across agents, eliminating heterogeneity in power budgets (i.e., 35W for each instead of 30W for 900 and 80W for 100). E-D is fair and avoids envy, but loses opportunities for system performance by allocating more power to those who benefit more.
- **Max-Welfare (M-W).** Allocate boosted rounds to maximize throughput. Sort agents by u, using an oracle, and allocate boosts to those with higher utility. Agents with low utility may starve.

We find that M-F performs much better than other fair policies—R-R, E-P, and E-D—yet provides sharing incentives and mitigates ex-post envy. M-F sees only

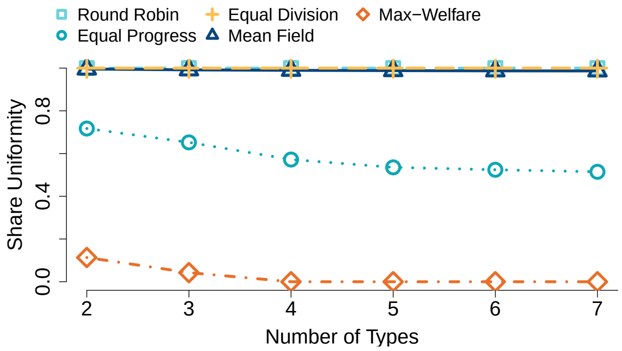


FIGURE 4.7: **Share Uniformity.** M-F, E-D, and R-R achieve uniform shares, M-W and E-P violate SI_R.

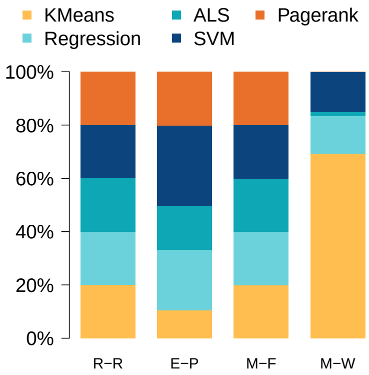


FIGURE 4.8: **Sharing Incentives.** Time in boosted rounds.

modest penalties relative to M-W, which provides an upper bound on performance that neglects fairness and starves low-throughput jobs.

4.7.1 Sharing Incentives

Allocations provide sharing incentives (SI_R) when agents receive the number of boosts they would have received under R-R. This definition is conservative. Equal time on boosted processors is sufficient, but not necessary, for SI_R. Agents given fewer boosts could still prefer M-F over R-R because M-F's boosts could deliver exceptionally high utility from successful signals while R-R's could be untimely.

We quantify SI_R with share uniformity, a metric calculated from allocated boosts over time.

$$\text{share uniformity} = \frac{\min\{\#\text{ boosted rounds}\}}{\max\{\#\text{ boosted rounds}\}}$$

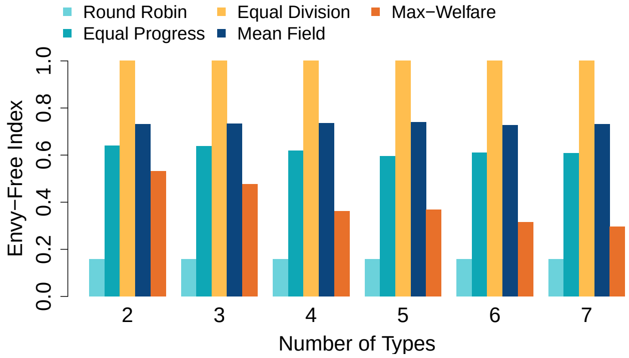


FIGURE 4.9: **Envy-free Index.** E-D achieves envy-freeness. M-F has high envy-free index. R-R, E-P, and M-W ignore envy.

Uniformity is the ratio of the minimum and maximum number of boosted rounds across agents, and its value is between 0 and 1. Larger values indicate stronger SI_{R}.

Figure 4.7 evaluates SI_{R} in terms of share uniformity. Uniformity is 1 for M-F and R-R as both allocate an equal number of boosts to each agent. Uniformity is 1 for E-D too as agents receive a 5W boost in every round.

In contrast, uniformity is much lower for E-P as it boosts stragglers and starves agents who make good progress without extra power. Similarly, uniformity is near 0 for M-W as it pursues performance by boosting agents who benefit most from extra power while starving the rest. Low uniformity due to starvation is likely in diverse populations with many workload types.

Figure 4.8 considers five representative workload types and their time shares for boosted processors. R-R provides SI_{R}, allocating boosts to agents in fixed rotation and guaranteeing equal time (*i.e.*, 20% each). M-F provides SI_{R} with similar time shares. Yet M-F is preferable as signals and tokens allow agents to request boosts when benefits are greatest, significantly improving performance – see Section 4.7.3.

Time shares reveal how E-P and M-W violate SI_{R} due to biases toward particular workload behaviors. E-P equalizes progress by favoring low-throughput workloads over high-throughput ones (e.g., `SVM` over `KMeans`). M-W maximizes system throughput by favoring workloads that benefit most and starving low-throughput ones (e.g., `KMeans` over `Pagerank`). These biases increase with workload heterogeneity.

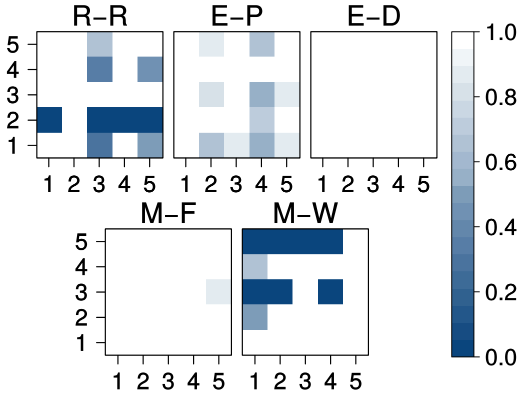


FIGURE 4.10: **Envy-free Index.** Darker colors mean greater envy. Five types include (1) `KMeans`, (2) `Regression`, (3) `ALS`, (4) `SVM`, (5) `Pagerank`.

4.7.2 Envy-Freeness

A mechanism is envy-free (EF_R) if no agent envies another's allocation sequence.[2] Agent i is envious when utility from its allocation is less than utility from another's. We use an index to measure agent i's envy toward agent j.

$$\text{EF index (pairwise)} = e_{ij} = \frac{u_i(x_i)}{\max(u_i(x_i), u_i(x_j))}$$

Larger indices correspond to less envy. If agent i envies agent j, then $u_i(x_i) < u_i(x_j)$ and $e_{ij} < 1$.

The index of agent i measures the greatest envy induced by any other agent.

$$\text{EF index (population)} = e_i = \min_j e_{ij} = \frac{u_i(x_i)}{\max_j\{u_i(x_j)\}}$$

Figure 4.9 evaluates EF indices for diverse agent populations. Figure 4.10 details EF indices for five representative workload types. A square at row i and column j indicates i's index towards j. Darker squares indicate greater envy.

M-F allocations induce little envy because the token system allows each agent to customize its sequence of boosts with strategic signals. Allocations tailored for

[2] Agent i's allocation is a sequence $x_i = (x_{i1}, \ldots, x_{ir})$ where $x_{ir} = 1$ if i receives a boost in round r and $x_{ir} = 0$ otherwise.

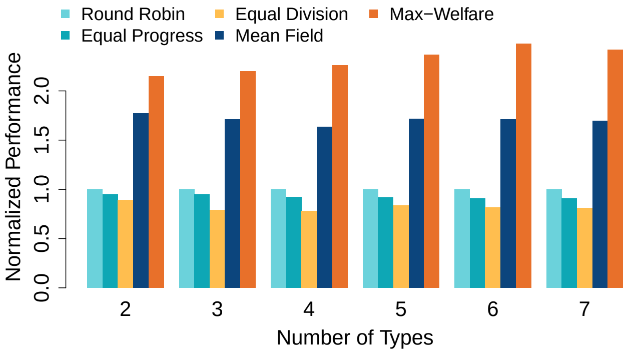


FIGURE 4.11: **Performance Evaluation.** Performance for 3 to 12 types. M-F outperforms R-R, E-D, and E-P and is comparable with M-W.

one agent are often unattractive to others, causing more agents to prefer their own allocations. Thus, M-F mitigates envy and reports an average EF index of 0.73. In other words, the average agent's utility from its allocation of boosts is within 73% of that from a competitor's. M-F is competitive with E-D, which avoids envy entirely by allocating the same 5W boost to every agent.

Strategic signals within the allocation game reduce envy among agents who do not receive boosts. If these agents did not signal, they did not need boosts and are not envious despite receiving nominal power budgets. If these agents signaled unsuccessfully, they must have already spent tokens for boosts in prior rounds. Failed signals induce little envy because swapping allocation sequences to get a power boost in the present would have required surrendering valuable boosts from the past.

Other baseline policies all induce substantial envy. R-R offers untimely boosts to indifferent agents (index=0.15). E-P boosts agents with poor progress who use extra power inefficiently (index=0.61). These policies induce envy in agents who would have gained more from boosts.

M-W boosts agents that benefit most but induces envy among many other agents (index=0.39). In Figure 4.10, M-W favors `KMeans`, making it envy-free (white row) while starving and inducing envy in others (dark rows). Envy worsens with increasing workload diversity.

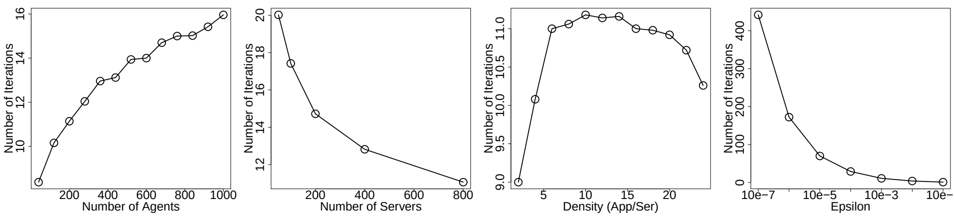


FIGURE 4.12: **Sensitivity.** (a) number of agents, (b) number of power boosts, (c) initial number of tokens.

4.7.3 Performance

The allocation game balances the pursuit sharing incentives and envy-freeness with performance. Although the game underperforms an approach that seeks throughput alone, it significantly outperforms other approaches to fairness.

We measure performance in terms of gains in job progress. Suppose that agent i receives allocation sequence $x_i = (x_{i1}, x_{i2}, \ldots)$ where $x_{ir} = 1$ if boosted in round r and $x_{ir} = 0$ otherwise. If boosted, progress improves by $u^B_{ir} - u^N_{ir}$, the difference in utilities under boosted and nominal power. These gains accumulate over time to determine performance.

$$p_i = \sum_r x_{ir} \left(u^B_{ir} - u^N_{ir} \right)$$

Figure 4.11 presents performance for diverse agent populations. M-F outperforms R-R by 1.7x, on average, by boosting agents when their benefits are greatest. Boosts are timely because agents optimize signaling strategies based on evolving utilities and competitive dynamics. In contrast, R-R performance suffers as agents boost in fixed rotation regardless of utilities. E-P performance also suffers as it diverts power to agents with inherently slow computation.

M-F also outperforms E-D by 2x, on average. E-D divides 35KW power budget among 1000 agents equally (*i.e.*, 35W per agent), which is equivalent to granting a 5W boost to every agent in every round. Unfortunately for performance, some workloads may not need extra power (e.g., memory-bound tasks) and boosts could have been profitably diverted to others (e.g., compute-intensive tasks).

M-F achieves 74% of M-W's performance, on average, a modest loss in exchange

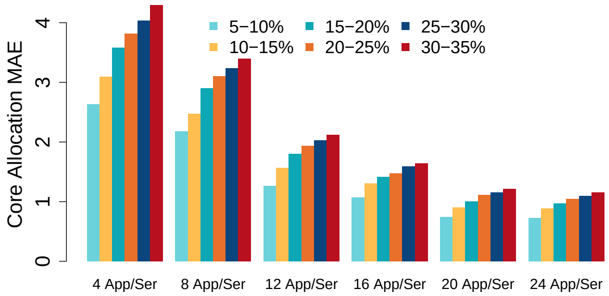


FIGURE 4.13: **Sensitivity to Interference.** Performance, share uniformity, and envy-free index are affected due to interference.

for fairness. M-F's largest losses arise when one type of agent benefits from boosts much more than others. In such cases, M-W favors the high-utility agents and starves others. Allocating boosts to low-utility agents, even for a small fraction of time, significantly degrade M-F's system throughput. And these degradations become worse as diversity increases.

4.7.4 Sensitivity to Game Parameters

Figure 4.12 assesses M-F's performance sensitivity to game parameters. Each study samples 15 pairs of workloads. For each pair, half the agent population runs the first workload and half runs the second. We measure performance, average across agents in the population, average across sampled workload pairs, and normalize to M-W's performance.

First, M-F performs better for larger populations. When the number of agents is small, the game is less likely to produce optimal strategies because mean field equilibria assume many agents. As the number of agents increases, threshold strategies approach optimal game play. This study varies the number of agents while ensuring enough boosts for 10% of the population.

Supporting more boosts increases M-F performance initially, but then produces diminishing returns. When boosts are scarce, agents with great need but few tokens may lose to agents with modest need but many tokens, which harms performance. Added boosts mitigate these rare outcomes and improve performance. But when

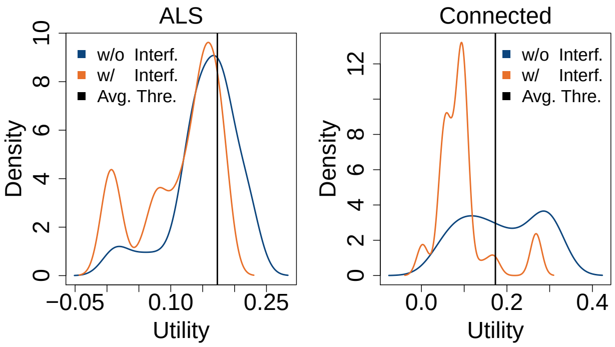


FIGURE 4.14: **Utility Distribution.** Utility with and without interference between `ALS` and `Connected`. Threshold is optimized assuming no interference and averaged across token holdings.

boosts are abundant, agents inefficiently lower thresholds and signal even when utilities are low.

Token holdings also affect signals and performance. As agents' average token holdings increase, they lower thresholds and signal more frequently. Because signals carry less information about agents' relative utilities, M-F's performance falls toward R-R's. This study varies the number of tokens in circulation for a fixed number of agents and boosts.

Finally, increasing the time per round increases M-F's performance initially, but then produces diminishing returns. Short rounds risk dividing a single workload phase and increasing correlation between utilities across rounds, which reduces the likelihood of optimal strategies from mean field analysis. In contrast, long rounds risk combining distinct phases and producing an uninformative average. When agents cannot differentiate power demands across rounds, they cannot signal strategically and M-F's performance suffers.

4.7.5 Sensitivity to Interference

The evaluation thus far assumes isolation between colocated workloads, but contention for shared resources (e.g., cache capacity and memory bandwidth) may affect the game's effectiveness. M-F optimizes threshold strategies according to profiles of jobs running alone. If those jobs actually colocate with others, profiles will less accurately capture utility and strategies will fall short of optimal responses to the competition from other jobs.

Figure 4.13 assesses the effect of interference on game outcomes. First, we optimize agents' strategies assuming no interference. Then, we evaluate those strategies with and without interference for nine representative workload pairs. Results, which are normalized to M-W's and averaged across workload pairs, show that performance, share uniformity, and the envy-free index decrease by 3.3%, 2.0%, and 31.6% respectively.

Figure 4.14 shows how interference degrades system outcomes weakening models of workload utility used to optimize game play. Interference shifts utility distributions leftward and causes thresholds that are optimal without interference to become conservative. Agents that lose more performance to contention (e.g., `Connected` versus `ALS`) will signal more conservatively.

Conservative signaling strategies induce significant envy without much harming other outcomes. Agents that signal less often will receive less timely boosts and envy others more often. Yet performance losses are modest as conservative strategies cause tokens to accumulate and signals in high-utility rounds to be successful more often. Furthermore, sharing incentives remain robust because the game's equal allocation and redistribution of tokens guarantee each agent its minimum share of boosts.

We can mitigate interference in several ways. First, we could reduce processor load. We present results on highly-loaded processors with hyper-threaded cores. When hyper-threading is disabled, interference harms envy-freeness by 18% instead of 32%. Second, we could deploy new microarchitectures that guarantee isolation for the last-level cache and memory channel [104, 105, 106, 107]. Finally, agents could continuously update their utility profiles and re-optimize their thresholds.

4.8 Related Work

Fairness. Fairness has become important for resource scheduling. Dominant resource fairness ensures game-theoretic desiderata, including SI and EF, when allocating homogeneous cores and memory capacity [8]. Resource elasticity fairness makes similar assurances when allocating cache capacity and memory bandwidth [1]. These mechanisms guarantee fairness in space when there are many more resources than users, whereas our token mechanism guarantees fairness in time.

In a related work, Wang et al. present XChange, a market with dynamic price discovery and wealth distribution, which balances throughput and fairness [108]. Although the token redistribution in our token mechanism and the wealth redistribution in XChange seem to be similar, they are fundamentally different for one main reason. The token redistribution happens every round when users receive their resources, whereas wealth redistribution in XChange happens only one time and before users receive their final allocations. In the token mechanism, as users receive their share of resources, they spend their tokens. Spent tokens are then redistributed among other users equally. In XChange, however, based on a heuristic metric, different wealth is assigned to different users. Users then use their wealth to bid on different resources.

For fairness in time, Craeynest et al. [109] schedule heterogeneous multi-cores to ensure equal-progress. In contrast, we study fairness from lenses of microeconomics and game theory. Adopting a similar approach for a different problem, Gorokh et al. [110] arbitrate access to a single item between many users in a repeated setting. They propose a repeated auction using artificial currencies to guarantee truthfulness and maximize efficiency.

Nesbit et al. propose a memory scheduler that employs fair queuing and provides sharing incentives [42]. Ghodsi et al. [111] generalize fair queuing to a multi-resource setting. In fair queuing, resources are allocated at packet-granularity—a link remains assigned to a flow until its entire packet is sent. Dividing time into slots and switching between flows at fixed intervals is not desirable. As a result, guaranteeing exact fair shares is difficult and fair queuing must be approximated through discrete packet scheduling decisions [112, 31].

Scrip Systems.

Friedman et al. [113] propose a scrip system for Peer-to-Peer (P2P) networks where users provide each other with file sharing services. The goal of the scrip systems is improve system performance while preventing agents to become free riders (*i.e.*benefit from the system without contributing to it). The authors prove the existence of a non-trivial Nash equilibrium at which homogeneous agents play a well-behaved strategy. Kash et al. [114] extend the scrip system for heterogeneous users and [115] extends this model by studying the effect of collusion.

Buttyan et al. [116] propose a scrip system to stimulate cooperation in ad hoc mobile networks. Considering a similar setting, Xu et al. [117] design a token system to incentivize self-interested users to relay other nodes traffic in autonomic wireless relay networks. Shen et al. [118] consider the same setting and propose a token exchange framework at which tokens are used to mitigate interference among users. Unlike the decentralized system models for P2P networks and cooperative routing, our work focuses on a centralized system for datacenter architectures that allocates items to users. Our token system is designed to ensure fairness while achieving high system performance.

Andrews et al. [119] consider the problem of scheduling a time-varying wireless channel between multiple users. They propose a token system to optimize system throughput subject to certain lower and upper throughput bounds for different users. Their token system is proposed for a settings where strategic behavior is not expected (*i.e.*users do not lie about their demands). Our token system, however, is designed for settings when self-interested users report their demands strategically to maximize their own utility.

Power and Heterogeneity Management. Multi-core schedulers steer tasks to the most efficient processor cores, but neglect fairness [120, 121, 122, 98]. Unlike previous works, we focus on game-theoretic notion of fairness while maximizing overall performance.

Fan et al. [2] study a computational sprinting game in which multi-core chips share a power supply and sprint independently. Similar to this chapter, [2] uses dynamic programming to find mean field equilibrium strategies for power boosts. However, this chapter differs from [2] as they pursue performance and system stability and we pursue fairness, defined by sharing incentives and envy-freeness. To ensure fairness, we design a game defined by tokens exchange rules. We study the distribution of tokens across agents and drive equilibrium strategies based on users' token holdings. We show in practice that this chapter's game satisfies repeated envy-freeness and sharing incentives and achieves high performance.

Through changes in p-states and clock throttling, power capping technologies enforce limits on servers' power consumption [123, 124]. Femal et al. [125] study a global power allocation mechanism that ensures a node is assigned a local power limit according to the performance of its workloads. Moreover, many hierarchical frameworks have been proposed to allocate power budgets dynamically between workloads [126, 61, 127]. Co-Con [128] uses a power control loop and a performance control loop to make adjustments on power and performance at the cluster level. PEGASUS [87] uses a feedback-based controller to dynamically assign power caps to the most latency critical workloads. Finally, VPM Tokens [129] manages power from virtual machines' perspective while considering global performance.

4.9 Conclusions

We present a new approach for fair resource management in dynamic systems. The token system provides game-theoretic desiderata while offering flexibility, which enhances performance by allocating resources to jobs in time periods that benefit performance most. We demonstrate a fair, repeated allocation game for heterogeneous processors that generalizes to other resources.

Future research could extend the allocation game in several dimensions. First, the game treats all agents equally and adding priorities is an open problem. One mechanism provides more tokens to agents with higher priorities. Another mechanism reduces the number of tokens required for a successful request. Extending the game theory for these extensions is non-trivial.

Second, the game could manage dynamic and variably sized power boosts. Moreover, the system could dynamically divide the power delivery unit's capacity to tune the definitions of nominal and boosted power budgets. Designing and adapting sprinting policies is an open research problem [130]. We could extend the game theory for varied degrees of heterogeneity.

Finally, the game manages a single resource type across time. Even for one resource, the token system advances the state-of-the-art in computational economics by pursuing game-theoretic desiderata in dynamic settings. Extending the system for multiple resource types over time is an open problem.

5

Amdahl's Law in the Datacenter Era: A Market for Fair Processor Allocation

5.1 Introduction

Shared computer systems present resource allocation challenges. Users and jobs, which vary in their demands and importance, must divide limited resources to balance competing performance, efficiency, and fairness objectives. Fairness is particularly relevant for non-profit systems in which users share capital and operating costs. Such systems often serve business units within a technology company or research groups within a university [131, 132, 133]. Allocations are determined by organizational priorities and service classes rather than explicit payments.

Systems determine users' shares with one of three mechanisms. With reservations, users request and pay for resources. Allocations depend on users' requests but are inefficient when requests are over-sized and resources are under-utilized [134, 135]. With priorities, allocations depend on users' computation and relative importance, exposing users to interference and non-deterministic performance [136, 132]. Finally, with entitlements, each user is guaranteed a minimum allocation and under-utilized resources are redistributed [136, 131, 32, 133]. Entitlements, unlike alternatives, provide isolation and efficiency.

Entitlements for datacenters differ from those for a server. Within a server, proportional share schedulers allocate divisible resources [136, 32]. In theory, the datacenter provides a similar abstraction—a warehouse-scale machine with logically divisible resources. In practice, however, resources are physically distributed across servers in ways that constrain allocation. Jobs are assigned to servers and resources are partitioned along server boundaries. Because processor allocations perform differently depending on which servers provide the cores, users often prefer specific allocations on specific servers.

We design a market mechanism that divides a user's datacenter-wide entitlement across the servers that run her jobs. Users receive budgets in proportion to their entitlements and bid for processor cores on each server. The market sets prices based on bids and users bid based on prices. The market's centerpiece is the Amdahl utility function, which we derive from Amdahl's Law to model the value of each server's cores and calculate bids [137, 22]. In equilibrium, all cores are allocated and allocations are optimal. This equilibrium is fair because budgets satisfy entitlements and performs well because bids shift more resources to more parallelizable workloads.

The market for processors offers several attractive properties. First, allocations incentivize sharing as each user always receives her entitlement and sometimes receives more. Second, allocations are Pareto-efficient, which means no other allocation can benefit one user without harming another. Third, the market is strategy-proof when the user population is large and competitive, which means no user can benefit by misreporting utility from processors.

The market has modest management overheads. Sampled profiles are sufficient to fit Amdahl's Law. Moreover, a market that is customized for processor allocation and Amdahl utility is computationally efficient. We derive closed-form equations to calculate bids that lead to a market equilibrium. In contrast, markets for generic utility functions can accommodate varied resources, from memory to power, but require expensive optimization and search to determine allocations [108, 138].

In this chapter, we co-design a utility function and market mechanism for processor allocation (Section 5.2). We estimate the utility function's parameter, the workload's parallelizable fraction, by inverting Amdahl's Law (Sections 5.3–5.4). We derive a procedure for calculating bids and allocations that produce a market equilibrium (Section 5.5). Finally, we find that equilibrium allocations satisfy entitlements and perform well. (Section 5.6).

5.2 Motivation and Overview

5.2.1 Public vs. Private Datacenters

The trend in public datacenters is toward a resource-as-a-service (RaaS) model [139]. In this model, datacenter providers deploy economic mechanisms to dynamically set prices for different resources [140, 141]. Unlike public datacenters, in many private datacenters, users share a non-profit server cluster and its capital and operating costs. Examples include an academic cluster that combines servers purchased by different research groups, or a cluster shared by different departments within a company.

In public datacenters, economic mechanisms use monetary transfers to align users' incentives with provider's objectives (*e.g.*, maximizing revenue while eliciting truthful reports). In private datacenters, however, monetary transfers between cluster managers and users are not desirable, and in some cases are not feasible or allowed. In such settings, aligning users' incentives with global system objectives becomes extremely challenging and requires new techniques.

When monetary transfers are not feasible, a common technique is to deploy virtual money in the form of scrip systems. Unlike real money, virtual money is not intrinsically valuable to users. As a result, classical economic mechanisms that work perfectly with real money, are not applicable with virtual money. Despite this fundamental challenge, scrip systems have been extensively deployed in many system settings. For instance, P2P systems have long used scrip systems to address the free-rider problem [114, 115]. In this chapter, we seek to explore efficient ways to use virtual money to ensure that users receive their entitlements.

5.2.2 Entitlements

Shared computer systems must allocate resources to satisfy entitlements, which specify each user's minimum allocation relative to other users'. Different entitlements could arise from differences in organizational priorities or users' contributions to shared resources. When an Internet services company colocates interactive and batch jobs, entitlements may specify more resources for online jobs to meet service targets. When users contribute funds to procure and operate a cluster, entitlements may specify shares in proportion to contributions to ensure fairness.

For decades, entitlements have been a basis for resource management. Henry designs the Unix Fair Share Scheduler to assign shares to users and mitigate nondeterminism in performance from the Unix priority scheduler [136]. Kay and Lauder define fairness in terms of users rather than processes, which mitigates strategic behavior during heavy system activity [131]. Waldspurger and Weihl propose lottery scheduling, which allocates resources probabilistically based on users' holdings of a virtual currency [32]. Randomization permits fine-grained shares that are fair and efficient.

Entitlements have several advantages. First, entitlements provide isolation by explicitly defining minimum shares, unlike priority-based mechanisms that allocate differently depending on user colocation and system activity. Second, entitlements are efficient. When a user requires less than her share, unused resources are redistributed to others. Redistribution incentivizes sharing by providing not only a minimum allocation but also the possibility of additional resources. Finally, entitlements mitigate strategic behavior by specifying shares for users, not jobs, such that no user gains resources by launching more jobs.

Entitlements are relevant for any user community that shares a non-profit system and its capital and operating costs. Early examples include high-performance computing systems [131, 142, 143]. Today's examples include academic and industrial datacenters. An academic cluster combines servers purchased by researchers who have preferred access to their own machines and common access to others' idle machines [144]. Microsoft uses tokens, a form of lottery scheduling, to specify and enforce shares [133]. Google does not use entitlements and, consequently, suffers from the same challenges as other priority schedulers, which cannot guarantee performance isolation between users [132, 136].

5.2.3 Processor Allocation

We require new entitlement mechanisms for datacenter processors because each user's allocation is distributed across multiple servers. Users may demand more cores on certain servers that run jobs with greater parallelism. But satisfying demands for specific servers while enforcing datacenter-wide entitlements is difficult. Moreover, simply allocating proportional shares in each server may violate entitlements depending how jobs are assigned to servers.

For example, three users have equal entitlements but varied demands for specific servers. Three servers–A, B, and C–each have 12 processor cores. User 1 demands 8 cores on A, 4 cores on B, and 0 cores on C, which we denote with vector (8, 4, 0). Users 2 and 3 have demand vectors of (0, 4, 8) and (8, 8, 8), respectively.

The approach that enforces proportional shares on each server violates entitlements. On each server, a user receives her demand or entitlement, whichever is smaller. When entitlement exceeds demand, excess cores are redistributed to other users on the server according to their relative entitlements. For example, the Fair Share Scheduler would allocate as follows:

$$\text{User } 1 \leftarrow (6_{\text{A}}, 4_{\text{B}}, 0_{\text{C}}) \,,$$

$$\text{User } 2 \leftarrow (0_{\text{A}}, 4_{\text{B}}, 6_{\text{C}}) \,,$$

$$\text{User } 3 \leftarrow (6_{\text{A}}, 4_{\text{B}}, 6_{\text{C}}) \,.$$

Because users 1 and 3 both demand 8 cores on A, they receive their 4-core entitlements and equally divide the remaining 4 cores. Across servers, users 1 and 2 receive 10 cores while user 3 receives 16, which satisfies entitlements in each server but violates them in aggregate. The equally entitled users should have received 12 cores each.

Alternatively, the system could relax entitlements within servers while preserving them across the datacenter. Users would start with their proportional shares distributed uniformly across servers (*i.e.*, 12 cores across 3 servers). Users would then trade according to their demands on each server.

$$\text{User } 1 \leftarrow (8_{\text{A}}, 4_{\text{B}}, 0_{\text{C}}) \,,$$

$$\text{User } 2 \leftarrow (0_{\text{A}}, 4_{\text{B}}, 8_{\text{C}}) \,,$$

$$\text{User } 3 \leftarrow (4_{\text{A}}, 4_{\text{B}}, 4_{\text{C}}) \,.$$

In the example, user 1 trades its 4 cores on C for user 2's 4 cores on A. Resulting allocations violate entitlements within each server but satisfy them in aggregate. Moreover, these allocations are efficient and match users' demands better. This example motivates a holistic trading algorithm that finds high-performance allocations subject to datacenter-wide entitlements.

5.2.4 Market Mechanisms

A market is a natural framework for trading cores in users' entitlements. Users spend their budgets on cores that are most beneficial. In effect, users trade their spare cores on one server for extra cores on others. The market sets prices for cores according to

Table 5.1: Workloads and Datasets

ID	Name	Application	Dataset (Size)
1	`Correlation`	Statistics	webspam2011 [146] (24GB)
2	`Decision Tree`	Classifier	webspam2011 (24GB)
3	`Fpgrowth`	Mining	wdc'12 [76] (1.4GB)
4	`Gradient Des.`	Classifier	webspam2011 (6GB)
5	`Kmeans`	Clustering	uscensus [74] (327MB)
6	`Linear Reg.`	Classifier	webspam2011 (24GB)
7	`Movie`	Recommender	movielens [75] (325MB)
8	`Naive Bayes`	Classifier	webspam2011 (6GB)
9	`SVM`	Classifier	webspam2011 (24GB)
10	`Page Rank`	Graph Proc.	wdc'12 [76] (5.3GB)
11	`Connected Cmp.`	Graph Proc.	wdc'12 (6GB)
12	`Triangle Cnt.`	Graph Proc.	wdc'12 (5.3GB)
13	`Blackscholes`	Finance	native
14	`Bodytrack`	Vision	native
15	`Canneal`	Engineering	native
16	`Dedup`	Storage	native
17	`Ferret`	Search	native
18	`Raytrace`	Visualization	native
19	`Streamcluster`	Data Mining	native
20	`Swaptions`	Finance	native
21	`Vips`	Media Proc.	native
22	`X264`	Media Proc.	native

server capacities and user demands. Given prices, users bid for servers' cores based on their jobs' demands. The market collects bids, sets new prices, and permits users to revise bids. This process repeats until prices converge to stationary values. When users' budgets are set in proportion to their datacenter-wide entitlements, the market guarantees proportional shares across the datacenter.

Bids require accurate models of performance given processor allocations on each server. The example in Section 5.2.3 assumes that demand and utility could be represented with a single number, a popular approach in systems research [8, 145, 40]. But to understand its limits, suppose a user demands four cores. Hidden in this demand is a significant implication: increasing an allocation by one core provides constant marginal returns to performance up to four cores and a fifth core provides no benefit. This assumption is unrealistic for many parallel workloads.

Table 5.2: Server Specification

Component	Specification
Processor	Intel Xeon CPU E5-2697 v2
Sockets	2 Sockets, NUMA Node
Cores	12 Cores per Socket, 2 Threads per Core
Cache	32 KB L1 ICache, 32 KB L1 DCache 256 KB L2 Cache, 32 MB L3 Cache
Memory	256 GB DRAM

5.2.5 Amdahl's Law and Karp-Flatt Metric

Amdahl's Law sets aside the constant marginal returns implied by a user's numerical request for cores. Instead, it models diminishing marginal returns as the number of cores increases [137, 22]. Amdahl's Law models execution time on one core, T_1, relative to the execution time on x cores, T_x. If fraction F of the computation is parallel, speedup is:

$$s_x = \frac{T_1}{T_x} = \frac{T_1}{(1-F)T_1 + T_1F/x} = \frac{x}{x(1-F)+F} \tag{5.1}$$

Computer architects use Amdahl's Law for first-order analysis of parallel speedup. The Law assumes the parallel fraction benefits linearly from additional cores and the serial fraction does not benefit. Because these assumptions hold to varying degrees in real workloads, architects often use Amdahl's Law to estimate upper bounds on speedups.

In this chapter, we use Amdahl's Law directly to assess utility from core allocations. We find that actual performance often tracks Amdahl's upper bound for modern datacenter workloads, which exhibit abundant, fine-grained parallelism and few serial bottlenecks. For example, Spark partitions jobs into many small tasks and caches data in memory to avoid expensive I/O [147, 53].

Using Amdahl's Law is challenging because the parallel fraction F is often unknown. Expert programmers rarely know exactly what fraction of their algorithm or code is parallel. Fortunately, we can measure speedup s_x and estimate F with the inverse of Amdahl's Law, which is known as the Karp-Flatt metric [148].

$$F = \left(1 - \frac{1}{s_x}\right)\left(1 - \frac{1}{x}\right)^{-1} \tag{5.2}$$

But for which processor count x should we measure speedup? When Amdahl's Law is perfectly accurate, the answer would not matter as measured speedups from varied x's would all produce the same estimate of F. In practice, Amdahl's Law is an approximation and estimates of F may vary with x.

5.2.6 Mechanism Overview

We design a two-part mechanism for allocating datacenter processors given users' entitlements. First, the mechanism requires a utility function (Section 5.4). We propose Amdahl utility, a new class of utility functions based on Amdahl's Law. We determine each user's Amdahl utility with new methods for profiling performance and estimating a workload's parallel fraction, the key parameter in the function.

Second, the mechanism requires a market (Section 5.5). We design a market to allocate processors when users are characterized by Amdahl utility functions. New utility functions require new bidding algorithms. Our algorithm calculates bids from workloads' parallel fractions, which are estimated when fitting Amdahl utility. Bids are calculated efficiently with closed-form equations.

The mechanism tightly integrates a new utility function that models performance and a new market that allocates cores. Its two parts are co-designed to quickly find the market equilibrium. In equilibrium, users perform no worse, and often better, than they would with their entitlements narrowly enforced in each server (Section 5.6).

5.3 Experimental Methodology

We construct Amdahl utility functions by profiling parallel workloads on physical machines. We measure speedups for varied core counts, use Karp-Flatt to estimate each workload's parallel fraction, and assess variance in those estimates.

Workloads. Table 5.1 summarizes our PARSEC [36] and Spark benchmarks [53] with their representative datasets. PARSEC benchmarks represent conventional, multi-threading whereas Spark applications represent datacenter-scale task parallelism.

Each Spark job is divided into stages and each stage has multiple tasks. The number of tasks in each stage usually depends on the size of the input data. The first stage typically reads and processes the input dataset. Given Spark's default 32MB block size, a 25GB dataset is partitioned into approximately 800 blocks. The run-time engine creates one task to read and process each block. It then schedules tasks on cores for parallel processing. We run Spark applications in standalone mode.

Physical Server Profiling. Table 5.2 describes the Xeon E5-2697-v2 nodes in our experiments. Each node has 24 cores on two chip-multiprocessors. The local disk holds workload data. We deploy Docker containers for resource isolation [149]. We use `cgroup` to allocate processor cores and memory to containers.

We measure parallel speedups to fit the Amdahl utility function. We profile execution on varied core counts using Linux `perf stat` for PARSEC and the run-time engine's event log for Spark. To efficiently determine how execution time scales with dataset size, we sample uniformly and randomly from original datasets to create smaller ones and construct simple, linear models.

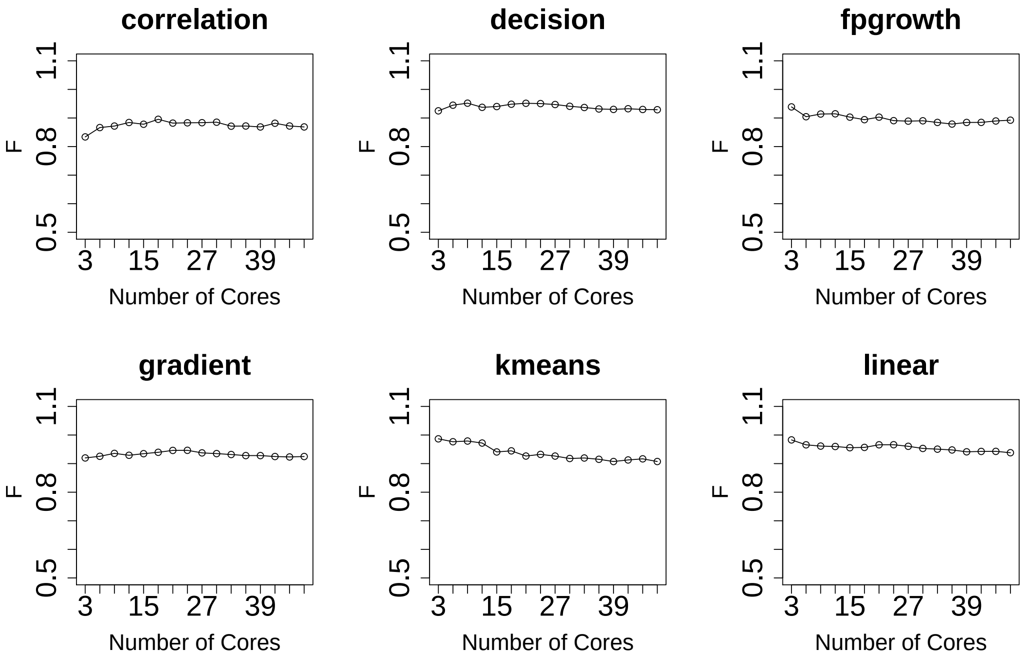


FIGURE 5.1: **Calculated Parallel Fraction.** F for representative Spark workloads as processor count varies.

5.4 Performance Model

Equation (5.2) is an idealized estimate of a workload's parallel fraction. In principle, inherent properties of the algorithm or code determine parallelizability. The processor count does not affect the parallel fraction but only determines how much of it is exploited for speedup. Note that Amdahl's Law assumes the parallel fraction is accelerated linearly with processor count.

$$F(x) = \left(1 - \frac{1}{s(x)}\right)\left(1 - \frac{1}{x}\right)^{-1} \tag{5.3}$$

In Equation (5.3), however, we describe the practical link between the workload's parallel fraction and system's processor count. We express parallel fraction F in terms of measured speedup $s(x)$ on x cores. The difference with Equation (5.2) is subtle but important. If the linearity assumption behind Amdahl's Law fails, the

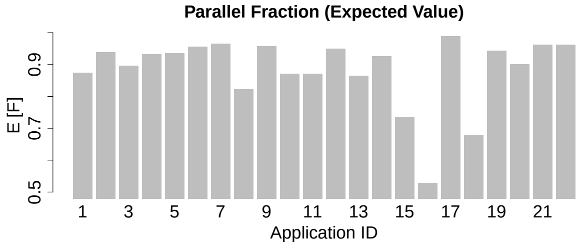


FIGURE 5.2: **Expected Parallel Fraction.** $\bar{F} = |x|^{-1} \sum_x F(x)$.

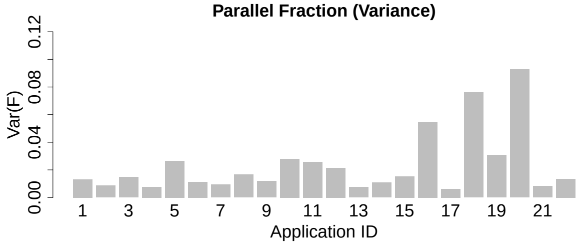


FIGURE 5.3: **Variance in Parallel Fraction.** $\mathrm{Var}(F) = |x|^{-1} \sum_x (F(x) - \bar{F})^2$. Lower variance indicates a better fit with Amdahl's Law.

number of processors deployed to profile speedup will affect the estimated parallel fraction.

Figure 5.1 empirically estimates the parallel fraction for representative workloads. We allocate x processors, measure speedup $s(x)$, and evaluate the Karp-Flatt equation for $F(x)$. The estimate is unaffected by processor count, indicating that Amdahl's Law accurately models speedup for most workloads. However, for some workloads, the estimate decreases as processor count increases, indicating parallelization overheads such as communication, shared locks, and task scheduling.

We report summary statistics for the estimated parallel fraction. Figure 5.2 presents average estimates from varied processor counts. The parallel fraction ranges from 0.55 to 0.99 for Spark and PARSEC workloads. Figure 5.3 presents variance in the estimate. For most workloads, variance is small and the Karp-Flatt analysis is useful. Estimates are consistent across processor counts and Amdahl's Law accurately models parallel speedups.

Although Karp-Flatt characterizes most workloads, it falls short when overheads increase with processor count. It is inaccurate for graph processing (e.g., `pagerank`, `connected components`, `triangle`) since tasks for different parts of the graph communicate more often as parallelism increases. Karp-Flatt is also inaccurate for computation on small datasets that require few tasks (e.g., `kmeans`'s 11 tasks) because adding processors rarely reduces latency and often increases scheduling overheads. Finally, it is inaccurate for workloads with intensive inter-thread communication (e.g., `dedup`) [36] because adding processors increases overheads.

5.4.1 Profiling Sampled Datasets

Estimating the parallel fraction requires profiling performance for varied processor counts. For efficiency, we reduce dataset sizes by sampling uniformly and randomly from the original dataset to create varied smaller ones. Sampled datasets are small enough that we can profile workloads' complete executions with all computational phases. Profiled speedups drive the Karp-Flatt analysis.

Sampled profiles reveal broader performance trends. Figure 5.4 shows how execution time scales linearly with dataset size when `correlation`, a representative workload, computes on 1GB to 6GB, 12GB, and 24GB of data.[1] Each line shows the model for a given processor count. Models are more accurate and data collection is faster when profiling computation on more processors (e.g., 48 cores). Venkataraman et al. similarly fit linear models using sampled datasets to predict performance on other datasets [150].

Although many workloads are well suited to linear performance models, some require polynomial models because their execution time scales quadratically with dataset size (e.g., QR decomposition). Although many datasets are amenable to

[1] Sampled datasets could be smaller than these as long as the number of partitions, which dictate the number of tasks, is greater than the number of processors. Otherwise, there is insufficient parallelism.

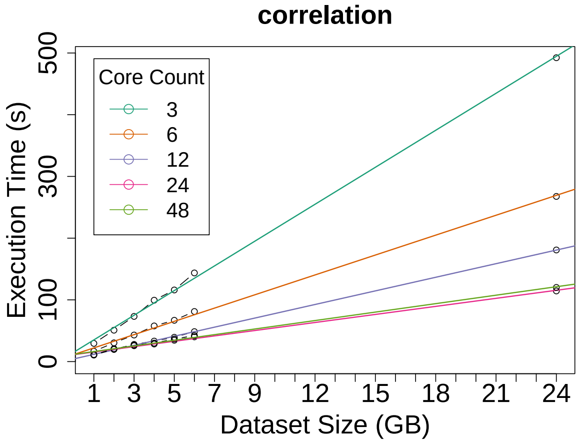


FIGURE 5.4: **Linear Model for Dataset Sampling.** Data shown for representative workload, `correlation`.

uniform sampling, skewed and irregular datasets (e.g., those for sparse graph analytics) require more sophisticated sampling.

5.4.2 Predicting Parallel Performance

Figure 5.5 combines Karp-Flatt and linear models to predict parallel performance. Karp-Flatt estimates parallel fraction from speedups (horizontal flow) and linear models estimate execution time from dataset size (vertical flow). Specifically, the procedure is:

- **Parallel Fraction F.** For each sampled dataset size d, estimate expected parallel fraction $\bar{F}_d$ for sampled core allocations. Report mean of $\bar{F}_d$'s.
- **Execution Time T.** For each sampled core allocation x, measure execution time T_x for sampled dataset sizes. Report linear model fitted to T_x's.

The procedure's outputs serve two purposes. First, we can estimate execution time for any processor count x and dataset size d from sparse profiles. Time measurements are scaled twice, by the linear model to account for the target dataset size and then by Amdahl's Law to account for the target processor count. Such scaling is

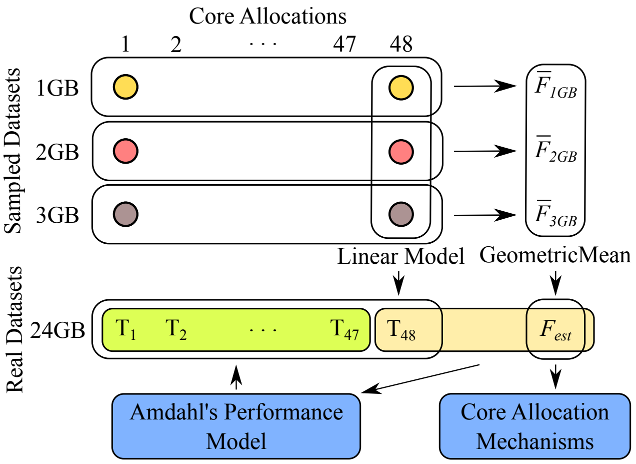


FIGURE 5.5: **Estimate Parallel Fraction.** Use linear models to estimate effect of dataset size. Use Karp-Flatt analysis to estimate effect of processor count.

accurate for varied parallel workloads—see Section 5.4.3. Second, we can construct Amdahl utility functions with estimated parallel fractions. Accurate functions enable markets that efficiently allocate processors to users according to entitlements—see Section 5.5.

5.4.3 Assessing Prediction Accuracy

We find that profiles on reduced inputs supply enough data for analysis. We can estimate workloads' parallel fractions, laying the foundation for markets with Amdahl utility functions. Moreover, we can estimate execution time for a variety of workload inputs and processor allocations.

Parallel Fraction. Figure 5.6 evaluates accuracy for the estimated parallel fraction. The estimated value is the geometric mean of Karp-Flatt analyses for multiple, sampled datasets. The measured value is the same but for the original dataset. For Spark, sampled datasets include 1GB to 6GB drawn randomly from the original dataset. For PARSEC, `simlarge` and `native` correspond to sampled and complete datasets, respectively.

Errors are small (*i.e.*, absolute accuracy) and estimates track measurements

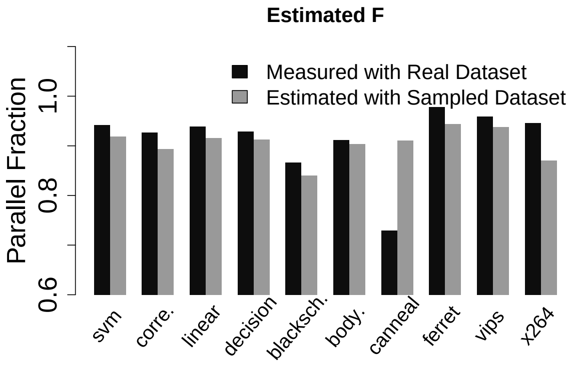


FIGURE 5.6: **Accuracy of Prediction.** Accuracy of predicted parallel fraction when using sampled datasets.

across workloads (*i.e.*, relative accuracy). Relative accuracy is particularly important for processor allocation. Karp-Flatt estimates the key parameter for Amdahl utility functions. And these utilities determine bids in the market for processors. Relative accuracy ensures more processors are allocated to users that benefit more, enhancing efficiency.

`Canneal` reports particularly high error because it is memory-intensive. Its memory bandwidth utilization on small datasets is not representative of that on larger datasets. When smaller datasets under-estimate bandwidth constraints, they overestimate speedups from additional processors. The estimated parallel fraction is much larger than the one measured on the full dataset.

Execution Time. Figure 5.7 evaluates accuracy for execution time. Good predictions rely on accurate scaling in two dimensions, by the linear model to account for the target dataset size and by Amdahl's Law to account for the target processor allocation. For the representative `Decision Tree` workload, we demonstrate accurate predictions for the target dataset and varied processor allocations.

Figure 5.8 broadens the evaluation to our workload suite. For each workload, a boxplot illustrates the range of errors when predicting execution time on varied processor allocations. Our models see 5-15% error, on average, and 30% error in

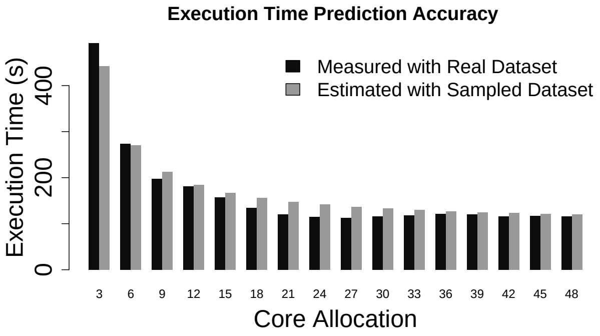


FIGURE 5.7: **Accuracy of Prediction.** Accuracy of predicted execution time given varied processor counts for `Decision Tree`.

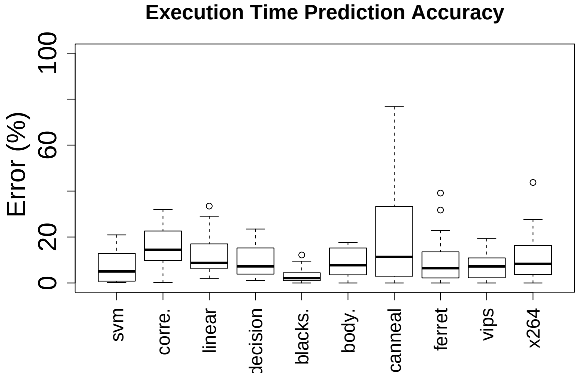


FIGURE 5.8: **Accuracy of Prediction.** Accuracy of predicted execution time for varied applications. Boxplots show distribution of errors given varied processor allocations.

the worst case. Cache- or memory-intensive applications (e.g., `canneal`) are poorly modeled as small, sampled datasets cause the predictor to over-estimate benefits from parallelism.

Although we evaluate execution time predictions, the broader goal is estimating parallelizability. The workload's parallel fraction concisely describes benefits from processors. Accurately estimating this fraction is a prerequisite for Amdahl utility functions. And these functions enable a market for processors.

5.5 Market Mechanism

We begin by formalizing the processor allocation problem. The system has n users and m servers that hold varied numbers of cores; server j has C_j cores. A user runs multiple jobs and each job has been assigned to a server. Note that we assume that the assignment of jobs to servers is fixed and job migration is expensive. Relaxing these assumption expands the state space of the problem and is an interesting future direction. In this paper, we focus on the problem of allocating the cores on each server given users' preferences and entitlements.

Our solution has two elements. First, we define the Amdahl utility function to describe users' preferences for cores. Second, we design a market in which users bid for cores according to utilities. We derive a new bidding algorithm to find the market equilibrium because there is no existing theory for Amdahl utility functions.

5.5.1 Amdahl Utility Function

Let f_{ij} denote the parallel fraction for user i's job on server j. From Amdahl's Law, allocating $x_{ij} \leq C_j$ cores to user i on server j produces speedup s_{ij}.

$$s_{ij}(x_{ij}) \;=\; \frac{x_{ij}}{f_i + (1 - f_i)x_{ij}}$$

Suppose that user i's job on server j completes w_{ij} units of work (*e.g.*, tasks) per unit time. We define the Amdahl utility function as user i's weighted average utility from cores across m servers.

$$u_i(x_i) \;=\; \frac{\sum_{j=1}^{m} w_{ij} s_{ij}(x_{ij})}{\sum_{j=1}^{m} w_{ij}} \tag{5.4}$$

Amdahl utility is consistent with architects' views of performance. Its parameters model important determinants of performance—exploitable parallelism (f) and work completed (w).

Although Amdahl utility resembles a weighted average of speedups, it actually measures normalized progress across multiple servers. Per unit time, a job completes w_{ij} units of work with one core and $w_{ij} s_{ij}(x_{ij})$ units with x_{ij} cores. Utility is total work completed normalized by that when allocated one core. Utility is one when the user receives one core per server as speedup is one on each server.

5.5.2 Market Model

We design a Fisher market with n participants described by Amdahl utility functions. Utility $u_i(x_i)$ describes user i's value from her allocation of $x_i = (x_{i1}, \ldots, x_{im})$ cores on each of m servers. After the market sets prices $p = (p_1, \ldots, p_m)$ for servers' cores, each user maximizes utility subject to her budget b_i, which is proportional to her entitlement.

$$\begin{aligned} \max \quad & u_i(x_i), \\ \text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} p_j \leq b_i. \end{aligned} \tag{5.5}$$

We illustrate market dynamics with an example. Suppose Alice and Bob share servers, C and D, each of which has ten cores. Alice runs `dedup` ($f = 53\%$) and `bodytrack` ($f = 93\%$) on C and D, respectively. Bob runs `x264` ($f = 96\%$) and `raytrace` ($f = 68\%$) on C and D, respectively. When Alice receives core allocation $x_A = (x_{AC},\ x_{AD})$ and Bob receives $x_B = (x_{BC},\ x_{BD})$, their utilities are as follows.[2]

$$\begin{aligned} u_{\text{Alice}} &= 0.5 \left(\frac{x_{AC}}{0.53 + 0.47 x_{AC}} + \frac{x_{AD}}{0.93 + 0.07 x_{AD}} \right), \\ u_{\text{Bob}} &= 0.5 \left(\frac{x_{BC}}{0.96 + 0.04 x_{BC}} + \frac{x_{BD}}{0.68 + 0.32 x_{BD}} \right). \end{aligned}$$

[2] Without loss of generality, this example assumes jobs complete one unit of work per unit of time (*i.e.*, $w_{ij} = 1$).

Suppose Alice and Bob have equal entitlements and budgets (*i.e.*, $b = 1$). When prices are $p = (0.04,\ 0.16)$, Alice determines her demand for processors as follows.

$$\begin{aligned} \max \quad & \frac{x_{AC}}{0.53 + 0.47 x_{AC}} + \frac{x_{AD}}{0.93 + 0.07 x_{AD}}, \\ \text{s.t.} \quad & 0.04\ x_{AC} + 0.16\ x_{AD} \leq 1. \end{aligned}$$

5.5.3 Market Equilibrium

In market equilibrium, all users receive their optimal allocations and there is no surplus or deficit of processors. Formally, price vector $p^* = (p^*_j)$ and allocation vector $x^* = (x^*_{ij})$ comprise an equilibrium under the following conditions.

1. **Market Clears.** All cores are allocated in each server j. Formally, $\sum_{i=1}^{n} x^*_{ij} = C_j$.

2. **Allocations are Optimal.** Allocation maximizes utility subject to budget for each user i. Formally, x^*_i solves Optimization (5.5) at prices p^*.

In the example, equilibrium prices are $p = (0.100, 0.099)$ for the two servers. At these prices, Alice receives $x_{\text{A}} = (1.34, 8.68)$ cores. She requests more processors on server D because her `bodytrack` computation has more parallelism. Bob receives $x_{\text{B}} = (8.66, 1.32)$.

Importantly, in a market equilibrium, users perform no worse than they would under their entitlements. We sketch the proof. First, under some mild conditions on utilities,[3] users exhaust their budgets in equilibrium. This, combined with the market-clearing condition, implies

$$\sum_j C_j p^*_j = B, \tag{5.6}$$

[3] For strictly monotonic, continuous and non-satiable utility functions, optimal allocation exhausts user's budget [41].

where B is the sum of users' budgets.

Next, since budgets are proportional to entitlements, user i is entitled to $x_{ij}^{ent} = (b_i/B)C_j$ cores on server j. Given Equation (5.6), it can be shown that users afford their entitlement allocation under equilibrium prices.

$$\sum_j x_{ij}^{ent}\ p_j^* = (b_i/B)\sum_j C_j p_j^* = b_i.$$

Thus, x_i^{ent} is a feasible solution of Optimization (5.5) for user i and price vector p^*. But since the equilibrium allocation x_i^* is the optimal solution,

$$u_i(x_i^*) \geq u_i(x_i^{entl}).$$

5.5.4 Finding the Market Equilibrium

Several cluster managers have designed markets and used a particular algorithm—proportional response dynamics (PRD)—to find their equilibria[151, 152, 153]. PRD is an iterative algorithm that uses simple and proportional updates for bids and prices [154, 155]. It is decentralized, inexpensive, and avoids optimization.

In each PRD iteration, users bid by dividing their budget across resources in proportion to the utility from them. Then, the market allocates by dividing resources across users in proportion to their bids for them. In response to allocations, users update bids. PRD ends when bids converge to stationary values.

The Fisher market equilibrium always exists because Amdahl utility is continuous and concave [156]. And we know that PRD converges to the equilibrium when utility functions have constant elasticity of substitution (CES) [157].[4]. But Amdahl utility is not a CES utility and existing PRD procedures do not apply.

We extend PRD to Fisher markets with Amdahl utilities, detailing the derivation here and summarizing the algorithm in Section 5.5.5. Our derivation re-writes

[4] The CES utility function has the form $u_i(x_i) = \sum_j (w_{ij}x_{ij})_i^{\rho}$.

the utility optimization problem and then assesses the effect on market equilibrium conditions (*i.e.*, market clears and allocations are optimal).

First, we turn the problem of finding the market equilibrium into a bidding problem. We re-write Optimization (5.5) with a new variable $b_{ij} = x_{ij}p_j$, which is user i's bid for cores on server j.

$$\begin{aligned} \text{max.} \quad & u_i(x_i), & & \quad (5.7) \\ \text{s.t.} \quad & x_{ij} = b_{ij}/p_j, & \forall j, & \\ & \sum_{j=1}^{m} b_{ij} \leq b_i, & & \\ & b_{ij} \geq 0, & \forall j. & \end{aligned}$$

The first constraint states that user i is allocated b_{ij}/p_j processors. The second constraint ensures that user i's bids across servers do not exceed her budget.

Next, we re-write the market-clearing condition in terms of equilibrium bids. The sum of users' allocations on server j must equal the server's capacity.

$$\sum_i x^*_{ij} = \sum_i b^*_{ij}/p^*_j = C_j.$$

The condition implies that, in equilibrium, server j's price is the sum of its bids divided by its capacity.

$$p^*_j = \sum_i b^*_{ij}/C_j. \tag{5.8}$$

Finally, the second equilibrium condition states that, given prices p^*, allocations x^* and bids b^* should be the optimal solution for Optimization (5.7). Using Lagrangian multipliers, for user i, there exists λ_i such that if $x^*_{ij} > 0$, then $\partial u_i/\partial x^*_{ij} = \lambda_i p^*_j$. Using algebra and the fact that $b^*_{ij} = x^*_{ij}p^*_j$, we conclude

$$\frac{{b^*_{ij}}^2}{{b^*_{ik}}^2} = \frac{f_{ij}\ p^*_j\ u^2_{ij}(x^*_{ij})}{f_{ik}\ p^*_k\ u^2_{ik}(x^*_{ik})}. \tag{5.9}$$

In equilibrium, b^*_{ij} must be proportional to $w_{ij}\ s_{ij}(x^*_{ij})$ and $\sqrt{f}$. Users bid more when the parallel fraction is larger and allocated cores provide larger speedups. Users also bid more when prices are higher, which indicates more competition for the server's cores.

5.5.5 Amdahl Bidding Procedure

Equations (5.8) and (5.9) define the Amdahl Bidding procedure. Users iteratively bid for servers' cores. In iteration t, server j's price is the sum of users' bids divided by its capacity.

$$p_j(t) = \sum_i b_{ij}(t)/C_j$$

Given these prices, user i's allocation of cores on server j is $x_{ij}(t) = b_{ij}(t)/p_j(t)$. She updates her bid by dividing her budget b_i across servers in proportion to utility from them.

$$\begin{aligned} b_{ij}(t+1) &= b_i U_{ij}(t)/U_i(t), \\ U_{ij}(t) &= \sqrt{f_{ij} p_j(t)}\ w_{ij}\ s_{ij}(x_{ij}(t)), \\ U_i(t) &= \sum_j U_{ij}(t). \end{aligned}$$

Updated bids lead to updated prices. The procedure continues until bids and prices converge to stationary values. We terminate when prices change by less than a small threshold ε. We can prove, using KKT conditions, that any fixed point of this procedure is a market equilibrium and vice versa.

5.6 Processor Allocation

We deploy workloads on physical servers (Section 5.3), profiling execution time on varied core allocations and datasets to fit Amdahl utility functions (Section 5.4).

Then, we construct a population of users that shares datacenter servers and run the market to allocate cores (Section 5.5). Finally, we measure allocation performance on physical servers.

User Populations. We construct a population of users and define key datacenter parameters. The number of users n is drawn uniformly from 40 to 1000, in increments of 80. Each user's budget and entitlement is drawn uniformly from 1 to 5. The number of servers m is defined in terms of a multiplier on the number of users. Specifically, $m = sn$ and s is drawn from $\{0.25,\ 0.5,\ 1,\ 2,\ 4\}$.

The workload density d is the maximum number of colocated jobs on a server. For each server, the number of jobs is drawn from $\{d/2, \ldots, d\}$ and the job itself is drawn from Table 5.1. Each job is randomly assigned to a user and every user runs at least one job. The competition for processors increases with density.

We construct 50 populations. Each population represents a different mix of workloads and their assignment to servers and users. We assess system performance and allocation outcomes for each population. Finally, we report data averaged across populations.

Rounding Allocations. Fair policies may produce fractional allocations. We use Hamilton's method to round fractional allocations to integral ones. Initially, we assign $\lfloor x_{ij} \rfloor$ cores to user i on server j. Then, we allocate any excess cores, one at a time, to users in descending order of their fractional parts.

Metrics. Let $\text{time}_{ij}(x_{ij})$ denote measured execution time for user i's job on server j when allocated x_{ij} cores. Let w_{ij} denote work completed per unit time when the job computes with one core. Because multiple cores reduce execution time and increase work rate, we measure a job's normalized progress as follows.

$$\text{JobProgress}_{ij}(x_{ij}) = w_{ij} \times \text{time}_{ij}(1)\ /\ \text{time}_{ij}(x_{ij})$$

User i distributes multiple jobs across datacenter servers and her aggregate progress is as follows.

$$\text{UserProgress}_i = \frac{\sum_j w_{ij} \times \text{time}_{ij}(1) \;/\; \text{time}_{ij}(x_{ij})}{\sum_j w_{ij}}$$

The numerator sums work completed across servers given the user's core allocation. The denominator sums work completed when the user receives only one core per server. This definition of progress matches the Amdahl utility function in Section 5.5. It also corresponds to the weighted speedup metric, which is used to study multi-threaded and multi-core systems [17, 158].

Finally, we define system progress as the weighted average of user progress. Weights reflect system priorities and are defined by users' budgets (*i.e.*, entitlements). Let b_i and B denote user i's budget and sum of all users' budgets, respectively. User i has weight b_i/B and system progress is as follows.

$$\text{SysProgress} = (1/B) \sum_i b_i \ \text{UserProgress}_i \tag{5.10}$$

5.6.1 Allocation Mechanisms

We evaluate our mechanism, which fits Amdahl utilities and invokes the **Amdahl Bidding (AB)** procedure. For each user, the mechanism instantiates an agent that bids for processors based on users' utilities and servers' prices. Bidding allows agents to shift cores from less parallelizable jobs to more parallelizable ones. Allocations are efficient and guarantee entitlements. We compare (AB) against several alternatives.

- **Greedy (G)** is a performance-centric mechanism that greedily allocates each core to the workload that yields the greatest speedup or progress. (G) uses an oracle to predict speedup for varied core allocations. This policy ignores entitlements when pursuing performance.

- **Upper-Bound (UB)** is a performance-centric mechanism that allocates cores to maximize system progress; see Equation (5.10). This policy favors users with larger budgets and entitlements when pursuing performance.

- **Proportional Sharing (PS)** is a fair mechanism that allocates each servers' cores in proportion to users' entitlements. If a user does not compute on a server, her share is reassigned to other users on that server in proportion to entitlements. (PS) enforces entitlements server by server but may violate them in aggregate. It neglects performance, ignoring users' unique demands for specific servers.

- **Best Response (BR)** is a market mechanism, like (AB), that balances fairness and performance. Users iteratively bid for resources, the market announces new prices, and users optimize bids with the interior point method [159]. Since Amdahl utilities are concave, the interior point method finds globally optimal bids in polynomial time.

In this paper, we focus on two important metrics: sharing incentives and performance. We compare against (UB) because its allocations achieve maximum system performance. We also compare against (PS) because its allocations provide sharing incentives by definition. We note that in today's datacenters, more sophisticated mechanisms are deployed. For instance, VMware DRS [160] implements distributed cluster-wide proportional shares. Although these mechanisms are considered to be more realistic baselines, they do not provide the game-theoretic properties we consider in this paper.

(AB) differs from (BR) in several regards. First, (AB) has lower overheads. (AB)'s bidding process evaluates closed-form equations to update bids given new prices whereas (BR)'s solves an optimization problem. In large systems, (BR) could incur prohibitively high overheads.

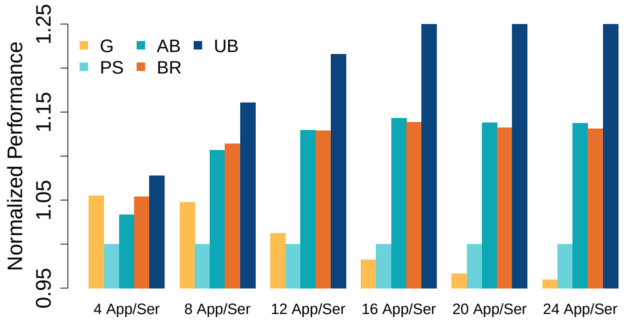


FIGURE 5.9: **Average System Performance.** Measured in terms of weighted system progress.

Second, (AB) is better suited to highly competitive systems. (AB) finds the Fisher market equilibrium when users are price-taking, which means users assume bids cannot change prices. This assumption is realistic in large systems. When many users share each server, an individual bid cannot significantly change the prices.

(BR), on the other hand, finds the Nash equilibrium when users are price-anticipating. Users realize their bids can change prices and that realization affects their bidding strategies. Individual bids are more likely to change prices in small systems.

5.6.2 System Performance

Figure 5.9 presents performance for varied allocation policies and workload densities. Performance is measured in terms of system progress and averaged over sampled user populations. Data is normalized relative to that from proportional sharing (PS). The figure illustrates the trade-offs between guaranteeing entitlements and pursuing progress.

(AB) outperforms (PS), the state-of-the-art in enforcing entitlements within each server. (PS) allocates cores in proportion to users' entitlements and redistributes any unused cores [136]. By focusing exclusively on entitlements, (PS) may allocate beyond the point of diminishing marginal returns from parallelism. Cores allocated to one user could have contributed more to another's progress.

(AB) achieves more than 90% of (UB)'s performance. (UB) allocates cores to maximize Equation (5.10). Its performance advantage increases with the competition for cores. When many workloads share the server, users rarely receive more than a few cores. (UB) allocates these scarce cores to users that contribute most to system progress, which improves performance disproportionately because the small allocations have not yet produced diminishing marginal returns. For example, the first three cores allocated to a workload has a larger impact than the next ten.

For similar reasons, (G)'s progress decreases with workload density. (G) allocates cores to the "wrong" user when the objective is system progress because entitlements are prominent in our measure of progress but ignored by the allocation policy. This effect gets worse when more users share each server. When cores are scarce and no workload reaches the point of diminishing returns, every core matters.

(AB) performs comparably with (BR), which has much higher implementation costs. (AB) finds the market equilibrium using closed-form equations and computational costs are trivial. In contrast, (BR) requires optimization with costs that scale with the number of users, workloads, and servers. Our (BR) implementation optimizes bids with the interior point method, but related studies use hill climbing [108].

(AB) produces a market equilibrium when users are price-taking whereas (BR) produces a Nash equilibrium when users are price-anticipating. Although (BR) is more robust to strategic behavior, price-anticipating users become price-taking ones when many users share the server and competition for resources is high. As density increases, (AB)'s market equilibrium approaches (BR)'s Nash equilibrium.

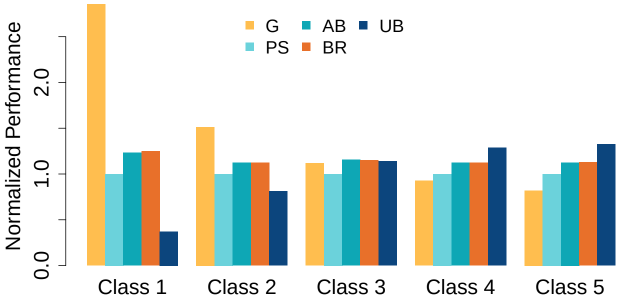


FIGURE 5.10: **Per-class Performance.** Measured in terms of user utility.

5.6.3 Entitlements and Performance

Figure 5.10 presents performance for users with varied entitlement classes. Budgets are proportional to class. For example, budgets for class 4 users are twice that of class 2 users. We report average performance over users within a class and normalize it by (PS)'s.

(G) neglects entitlements and disadvantages high-class users relative to (PS), which satisfies entitlements. In contrast, (UB) favors high-class users because their progress is weighted more heavily in Equation (5.10). Thus, performance-centric policies benefit high-class users while harming low-class ones or vice versa.

(AB) and (BR) guarantee entitlements for all classes. Users in all classes make similar progress because they have the budget to afford entitled allocations. We sketch the proof for (AB) in Section 5.5 and a similar one applies to (BR). Figure 5.10 presents the same finding empirically.

Moreover, (AB) and (BR) outperform (PS). Dividing budgets into bids for cores on specific servers is equivalent to trading cores on servers running less parallel jobs in return for cores on servers with more parallelism. These trades cause (AB) and (BR) to outperform (PS), which enforces proportional shares independently on each server.

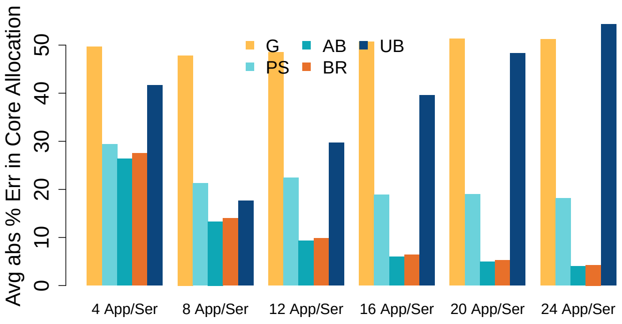


FIGURE 5.11: **Mean Absolute Percentage Error.** MAPE of core allocations under different allocation policies and global core entitlements.

5.6.4 Entitlements and Allocations

Figure 5.11 compare allocations against datacenter-wide entitlements by reporting the Mean Absolute Percentage Error (MAPE). Error is high, regardless of policy, when users run jobs on a few servers. Each user's allocated cores are drawn from the servers she computes on. Computing on fewer servers constrain allocation and make satisfying entitlements more difficult.

(G) and (UB)'s errors are large because they disregard entitlements. Under (G), users who have similar utilities receive similar core allocations despite reporting different entitlements. Under (UB), users that contribute more to system progress receive more cores, especially when density is high and cores are scarce.

(PS)'s errors are significant due to its proportional shares within each server. Users receive no more than their share on each server that runs their jobs, and users receive no compensating credit for servers that they do not use. This effect, first seen in Section 5.2, explains the gap between (PS)'s allocations and entitlements. In theory, (PS)'s errors fall to zero when users run jobs on every server in the system, but this scenario is practically impossible.

(AB) and (BR) address (PS)'s challenges by permitting users to shift and trade entitlements across servers even in violation of proportional shares within servers.

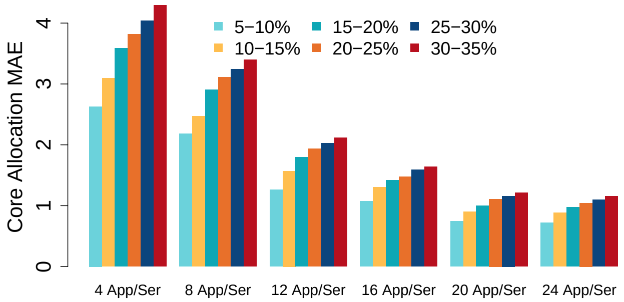


FIGURE 5.12: **Mean Abs Error.** MAE in core allocation due to over-estimation of $\bar{F}$.

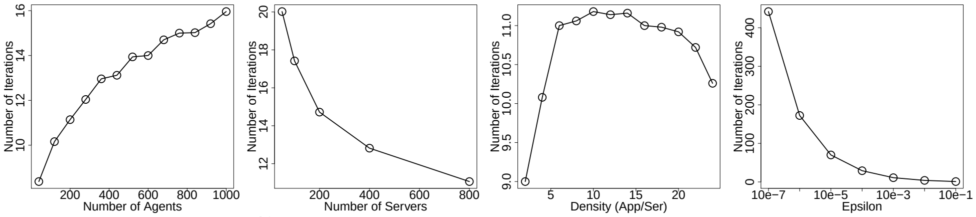


FIGURE 5.13: **Convergence Rate.** Rate of convergence for (AB).

The ability to trade freely increases with workload density. Although (AB) and (BR) report similarly low errors, their allocations are different.

Suppose a user runs two jobs, one on server C without co-runners and another on server D with multiple co-runners. Under (AB), the user is price-taking and divides her budget between servers. Under (BR), the user anticipates the effect of her bid on C's price, assigns a small fraction of her budget to C, and assigns the rest to D. (BR)'s strategic bids do not affect the user's allocation on C. But they do increase her allocation on D relative to what she would have received from (AB).

5.6.5 Interference Sensitivity

We estimate parallel fraction $\bar{F}$ by profiling workloads in isolation. However, workloads in real systems see interference from colocated computation. Colocation degrades performance, which implies smaller speedups from parallelism and smaller effective values for $\bar{F}$. By profiling workloads in isolation, we may over-estimate $\bar{F}$.

We assess (AB)'s sensitivity to colocation. We select a random user and reduce her jobs' parallel fractions by some percentage to reflect contention and corresponding estimation error for $\bar{F}$. In chip multiprocessors, competition for shared cache and memory typically degrades performance by 5 to 15% [161]. Finally, we compare market allocations when using original and adjusted estimates.

Figure 5.12 shows that over-estimating $\bar{F}$ may cause users to bid more and receive larger allocations. However, because contention causes the user to over-estimate $\bar{F}$ for all of her jobs, the net effect on how she divides her budget across jobs is small. For moderate workload densities, over-estimating $\bar{F}$ by 5 to 15% shifts an allocation by one or two cores.

5.6.6 Overheads

Finding equilibrium allocations is computationally efficient and requires 12.35ms, an average over 600 measurements. In each iteration, users spend 0.10ms updating bids and the market spends 0.85ms updating prices and checking the termination condition. Users communicate with the market and round-trip network delay ranges from 0.20 to 0.30ms.

After prices converge, often within ten iterations, the market distributes equilibrium bids to servers. Each server calculates equilibrium allocations, in parallel, with an overhead of 0.35ms. Of this time, 0.3ms is needed to receive bids and 0.05ms is needed to calculate and round fractional allocations.[5]

In BR, users spend on average 22× more time to update their bids than they do in AB. These overheads are problematic when architects opt for a centralized implementation of the market to avoid congesting networks with thousands of bidding messages. In centralized implementations, BR's procedure for updating bids produces prohibitively high overheads as network communication and calculating final allocations becomes a smaller share of total overhead and updating bids becomes a

[5] $12.35\text{ms} = 10\times(0.10\text{ms} + 0.85\text{ms} + 0.25\text{ms}) + 0.35\text{ms}$.

larger share.

Figure 5.13 shows how the number of iterations depends on system parameters. First, overhead increases with the user population size. As more users bid, the market requires more time to find stationary prices. Second, more servers implies more jobs per user and smaller shares of the budget for each job. Smaller bids cause smaller price updates, which lowers overheads.

Third, workload density affects overheads in non-monotonic ways. (AB) converges faster when workload density is low and only a few users bid for each server. As density increases, the number of bidders and overheads increases. But even further increases in density imply the system has users with many jobs, each with a small share of the budget and small bids that reduce overheads.

5.7 Related Work

Numerous studies cast hardware management as a market problem in which users bid for resources [108, 162]. XChange, the study that inspires our best-response (BR) baseline, is a market for allocating cache capacity, memory bandwidth, and power in a chip multiprocessor [108]. XChange supports piecewise-linear models that can take any shape and thus support more resource types. But the flexible models require search heuristics, which may get caught in local optima, to find the bids and prices that produce a Nash equilibrium. In contrast, we define Amdahl utility functions and propose a closed-form bidding mechanism that guarantees a market equilibrium.

Other studies explore game theory for systems management. Many researchers use the Leontief utility function to allocate datacenter cores and memory [8, 163, 40]. Zahedi et al. use the Cobb-Douglas utility function to allocate cache capacity and memory bandwidth [1]. Both studies guarantee game-theoretic desiderata such as sharing incentives, envy-freeness, Pareto efficiency, and strategy-proofness. The Fisher market in our study generalizes sharing incentives with entitlements, guarantees Pareto efficiency with its market equilibrium, and is strategy-proof for a large user population.

Our analysis compares two solution concepts—market and Nash equilibria—but others are relevant to systems and architecture. For datacenter power management, Fan et al. study mean field equilibria in which best responses are optimized against statistical expectations of competitors' actions [2]. For workload colocation in servers, Llull et al. study stable matches, a solution concept for cooperative games in which users' interactions determine a shared outcome [161].

Cloud infrastructure providers often require reservations, asking users to specify their desired resources such as the number of cores, amount of memory, and number of virtual machines. For example, Hindman et al. implement a request-grant abstraction among heterogeneous parallel frameworks [66]. Such systems rely on users to report resource usage, introducing opportunities for strategic action.

Finally, Gulati et. al enforce entitlements in a virtualized environment with VMware DRS, which allocates processors and memory to virtual machines across a distributed cluster [160]. Allocations are determined based on entitlements and demands based on actual consumption, which is vulnerable to manipulation and users' strategic behaviors. In contrast, we use Amdahl utility functions and Karp-Flatt metric to estimate users' demands on each server. The Amdahl bidding mechanism is theoretically strategy-proof in large systems.

5.8 Conclusion and Future Work

We introduce the Amdahl utility function, which concisely models user value from processor core allocations. We present a profiling framework that operationalizes Amdahl's Law, using its inverse—the Karp-Flat metric—to estimate the parallelizable fraction of a workload. Finally, we design a market mechanism that uses Amdahl utilities and a novel bidding procedure to allocate processors. Allocations ensure entitlements in a shared datacenter.

We design and evaluate the mechanism for processor cores. In the future, we will look beyond a single resource type. Amdahl's Law has been extended for heterogeneous core and can be generalized to reason about the diminishing returns from allocations of any hardware resource [22].

6

Dynamic Proportional Sharing: A Game-Theoretic Approach

6.1 Introduction

Shared systems are defined by the competition for resources between strategic agents. In this chapter, we consider a community of agents who share a non-profit system and its capital and operating costs. Sharing increases system utilization and amortizes its costs over more computation [6]. Examples include supercomputers for scientific computing [164], datacenters for Internet services [133, 132], and clusters for academic research [165, 144]. Note, however, that our focus excludes systems in which agents explicitly pay for time on shared computational resources (i.e., infrastructure-as-a-service).

Shared systems ensure fairness by allocating resources proportionally to entitlements, which specify each agent's share of system resources relative to others [136, 131, 32]. Entitlements are dictated by exogenous factors such as agents' contributions to the shared system or priorities within the organization. A dynamic allocation mechanism should ensure agents' entitlements across time while assigning resources to computational stages that benefit most.

Guaranteeing entitlements and redistributing under-utilized resources are difficult when agents are strategic. The allocation mechanism does not know and must extract agents' utilities, which are private information. Strategic agents act selfishly to pursue their own objectives. Agents will determine whether misreporting demands can improve their performance even at the expense of others in the system. For example, an agent is likely to over-report her demand in the current time period to obtain more resources, unless doing so leads to a reduction in the resources allocated to her in later periods.

We seek allocation mechanisms that satisfy strategy-proofness (SP), which ensures that no agent benefits by misreporting her demand for resources. Strategy-proofness is a key feature contributing to efficiency as it allows the mechanism to optimize system performance according to agents' true utilities. Without SP, agents' reports may not represent their true utility and allocating based on reported demands may not produce any meaningful performance guarantee. Moreover, strategy-proof mechanisms reduce the cognitive load on agents by eliminating the need to optimally construct resource demands or preemptively respond to misreports by other agents in the system.

Strategy-proofness is complemented by sharing incentives (SI), which ensures that agents perform at least as well as they would have by not participating in the allocation mechanism (i.e., using their own resources as a smaller, private system). With sharing incentives, agents would willingly federate their resources and manage them according to the commonly agreed upon policy. A mechanism that statically enforces entitlements in every time period satisfies strategy-proofness and sharing incentives but its efficiency is poor and fails to realize the advantages of dynamic sharing across time.

In this chapter, we focus on three fundamental game-theoretic desiderata: sharing incentives, strategy-proofness, and efficiency. We consider agents who derive high utility per unit of resource up until some amount of resource allocation (i.e., their demand) and derive low utility beyond that allocation. The high-low formulation is appropriate for varied resources such as processor cores, cache and memory capacity, or virtual machines in a datacenter. For example, an agent could derive high utility when additional processors permit her to dequeue more tasks from a highly critical job. Once the job's queue is empty, she derives low utility from using additional processors to replicate tasks, which guards against stragglers or failures. In another example, an agent that is allocated more power can turn on more processors, each of which provides high utility from task parallelism. Once the agent exhausts her job's parallelism, it can use additional power to boost processor voltage and frequency for lower, non-zero utility.

We propose allocation mechanisms for dynamic proportional sharing to address limitations in existing approaches. We begin by proving that policies used in state-of-the-art schedulers [166, 167, 168] fail to satisfy SP or SI. We then propose two alternative mechanisms. First, as our main contribution, we propose the flexible lending mechanism to satisfy SP, guarantee at least 50% of SI performance, and provide an asymptotic efficiency guarantee. The mechanism uses tokens to enable these theoretical guarantees. In practice, our simulations show that performance is comparable to that of state-of-the-art mechanisms and achieves 98% of SI performance, much better than the lower bound. Second, for situations where SI is a hard constraint, we propose the T-period mechanism to satisfy SP and SI while still outperforming static allocations.

6.2 Preliminaries

Consider a dynamic system with n agents and R discrete rounds. Agent i contributes $e_i > 0$ units of a resource at each round, which we refer to as her *endowment*. In

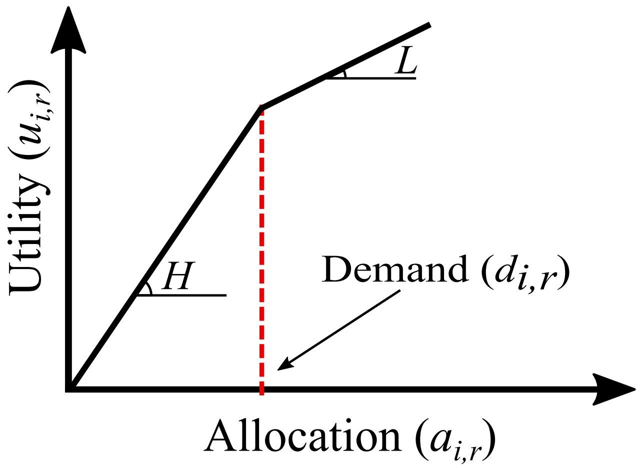


FIGURE 6.1: **Users Utility.** A user derives high utility from resources up to her demand and derives low utility from resources beyond her demand.

other words, e_i is agent i's contribution to the federated system, which does not vary over time. Let $[n] = \{1, \ldots, n\}$ and $E = \sum_{i \in [n]} e_i$ denote the total number of units to be allocated at each round. At round r, agent i has a true demand of $d_{i,r} \geq 0$ units and reports a demand of $d'_{i,r} \geq 0$. Let $\mathbf{d}'_i = (d'_{i,1}, \ldots, d'_{i,R})$ denote the vector of agent i's reports, and $\mathbf{d}'_{-i}$ denote the reports of all agents other than i.

A dynamic allocation mechanism M assigns each agent an allocation $a^M_{i,r}(\mathbf{d}'_i, \mathbf{d}'_{-i})$ using only information from the first r entries in the demand vectors. We will often write simply $a_{i,r}$ when the exact mechanism and the demands are clear from context. Let $\mathbf{a_i^M}(\mathbf{d}'_i, \mathbf{d}'_{-i})$, often simply $\mathbf{a_i}$, denote the vector of agent i's allocations. Agents have high (H) utility per resource up to their demand, and low (L) utility per resource that exceeds their demand. Formally, the utility of agent i at round r for $a_{i,r}$ units is denoted by $u_{i,r}(a_{i,r})$ and modeled as the following.

$$u_{i,r}(a_{i,r}) = \begin{cases} a_{i,r}H & \text{if } a_{i,r} \leq d_{i,r}, \\ d_{i,r}H + (a_{i,r} - d_{i,r})L & \text{if } a_{i,r} > d_{i,r}. \end{cases}$$

Figure 6.1 shows $u_{i,r}$ for user i with demand $d_{i,r}$ at round r. For simplicity, we assume H and L are the same for all agents, but all our results extend to the case where agents have different values of H and L (with the exception of Section 6.5.5).

While resources and demands are discrete, we allow the allocations $a_{i,r}$ to be real-valued. Real-valued allocations can be thought of as probabilistic—the realized allocation is a random allocation where agent i is allocated $a_{i,r}$ resources in expectation, which is always possible as a result of the Birkhoff-von Neumann theorem [169]. Agent i's overall utility after R rounds for allocation $\mathbf{a_i}$ is calculated additively as follows.

$$U_{i,R}(\mathbf{a_i}) = \sum_{r=1}^{R} u_{i,r}(a_{i,r}).$$

We do not consider discounting for simplicity of presentation, but our mechanisms readily extend to the case where agents discount their utilities over time.

In this chapter, we focus on three main properties: strategy-proofness, sharing incentives, and efficiency. First, strategy-proofness says that agents never benefit from lying about their demands. In other words, agent i's utility decreases if she reports $\mathbf{d'_i} \neq \mathbf{d_i}$.

Definition 1. *Mechanism M satisfies* strategy-proofness (SP) *if*

$$U_{i,R}(\mathbf{a_i^M}(\mathbf{d_i}, \mathbf{d'_{-i}})) \geq U_{i,R}(\mathbf{a_i^M}(\mathbf{d'_i}, \mathbf{d'_{-i}})) \qquad \forall i, \forall R, \forall \mathbf{d_i}, \forall \mathbf{d'_i}, \text{ and } \forall \mathbf{d'_{-i}}.$$

Next, sharing incentives says that by participating in the mechanism, agents receive at least the utility they would have received by not participating.

Definition 2. *Mechanism M satisfies* sharing incentives (SI) *if*

$$U_{i,R}(\mathbf{a_i^M}(\mathbf{d_i}, \mathbf{d'_{-i}})) \geq U_{i,R}(\mathbf{e_i}) \qquad \forall i, \forall R, \forall \mathbf{d_i}, \text{ and } \forall \mathbf{d'_{-i}}.$$

We also define a relaxed notion of α-sharing incentives, which says that every agent gets at least an α fraction of the utility that she would have received without taking part in the mechanism. Note that 1-SI is equivalent to SI.

Definition 3. *Mechanism M satisfies* α-SI *if*

$$U_{i,R}(\mathbf{a_i^M}(\mathbf{d_i}, \mathbf{d'_{-i}})) \geq \alpha\ U_{i,R}(\mathbf{e_i}) \qquad \forall i, \forall R, \forall \mathbf{d_i},\ \textit{and}\ \forall \mathbf{d'_{-i}}.$$

Finally, efficiency says that all resources should be allocated, and an agent with L valuation should never receive a resource while there are agents with H valuation for that resource.

Definition 4. *Mechanism M satisfies* efficiency *if*

$$\sum_{i \in [n]} a_{i,r}^M = E,$$

and if $a_{i,r}^M > d'_{i,r}$ for some agent i and round r, then $a_{j,r}^M \geq d'_{j,r}$ for all agents.

Note that efficiency is relative to the agents' *reports*, not their actual valuations, which are hidden from the mechanism. Therefore, in situations where agents lie about their valuations, it is possible that even an efficient mechanism allocates a unit inefficiently with respect to the actual valuations. With this in mind, there is little value in a mechanism that is efficient but not SP. Similarly, if a mechanism does not satisfy SI, then agents may not want to participate in it. So an efficient mechanism that does not satisfy SI may not actually exhibit efficiency gains in practice because agents choose not to participate. In some contexts, SI may not be of concern because agents are forced to participate or are willing to risk participation if gains are likely large and losses are likely small.

For readability, some proofs are omitted and appear in the appendix.

6.3 Existing Mechanisms

In this section, we focus on the (weighted) max-min fairness policy, which is one of the most widely used policies in computing systems. It is deployed in many state-of-the-art datacenter schedulers such as the Hadoop Fair Scheduler [166], Hadoop Capacity

Scheduler [167] and Spark Dynamic Allocator [168]. And it has been extensively studied in the literature [8, 170, 171].

A dynamic allocation mechanism could deploy the max-min policy for two different objectives: maximizing the minimum accumulated allocations up to a round, or maximizing the minimum allocation at each round, independently of previous rounds. We call the first mechanism *Dynamic Max-Min (DMM)* and the second mechanism *Static Max-Min (SMM)*. First, at each round r, DMM selects the allocation that maximizes $\min_i \sum_{r'=1}^{r} a_{i,r'}/e_i$, the minimum weighted cumulative allocation; subject to this, it maximizes the second lowest weighted cumulative allocation, and so on. This maximization is subject to the constraint that no resource is allocated to an agent with low valuation as long as there are agents with high valuation.

Second, at each round r, SMM selects the allocation that maximizes $\min_i a_{i,r}/e_i$, the minimum weighted allocation at that round; subject to this, it maximizes the second lowest weighted allocation, and so on. This maximization is also subject to the constraint that no resource is allocated to an agent with low valuation as long as there are agents with high valuation. Under SMM, agents are guaranteed to receive their demands as long as they are less than or equal to their endowment. Agents with demands higher than their endowments receive extra resources from agents with demands lower than their endowments. Unlike DMM, SMM allocates resources locally at round r, regardless of agents' allocations prior to round r.

In the rest of this section, we study properties of these two mechanisms. In particular, we focus on three properties: strategy-proofness, sharing incentives, and efficiency. We examine whether the existing mechanisms satisfy these properties for the special case when $L = 0$ and for the general case when $L > 0$.

6.3.1 Properties of Mechanisms for $\mathbf{L} = \mathbf{0}$

When $L = 0$, one might think that agents do not have any incentive to misreport their demands. However, we show that DMM fails to satisfy SI and SP.

Theorem 5. *Dynamic max-min mechanism violates sharing incentives, even when* $L = 0$*.*

Proof. Suppose that $R = 10$ and there are three agents, each with $e_i = 3$. For all rounds $r \neq 10$, the demands are $d_{1,r} = 1$, $d_{2,r} = 2$, and $d_{3,r} = 6$. For rounds $r = 1, \ldots, 9$, each agent is allocated exactly her demand. After round 9, utilities for agents 1, 2 and 3 are $9H$, $18H$ and $54H$, respectively. At round 10, demands are $d_{1,10} = 9$, $d_{2,10} = 9$, and $d_{3,10} = 6$. DMM allocates all 9 units to agent 1, which maximizes the minimum weighted cumulative allocation. Consider agent 2. Under DMM, agent 2's allocation is $a_{2,r} = 2$ for all $r \neq 10$ and $a_{2,10} = 0$. If she had not participated in the mechanism, then she would have obtained the same utility in each round $r \neq 10$, but a strictly higher utility in round $r = 10$. □

Theorem 6. *Dynamic max-min mechanism violates strategy-proofness, even when* $L = 0$ *[172].*

Proof. Consider three agents with equal endowments $m_1 = m_2 = m_3 = 1$ sharing three units of a resource for three rounds. The demand of agent 1 is 3 for all three rounds. Agent 2's demand is 3 for rounds 1 and 3 and 0 for round 2. And agent 3 has a demand of 3 for round 2 and 0 for rounds 1 and 3. Agent 1 achieves utility of $3.375H$ by truthful reporting. If agent 1 misreports 0 for round 1, her utility would increase to $3.75H$. □

Since DMM does not satisfy SP, it cannot guarantee any meaningful notion of efficiency, as explained in Section 6.2. Next, we show that SMM satisfies SI, SP, and efficiency.

Theorem 7. *Static max-min mechanism satisfies strategy-proofness, sharing incentives, and efficiency when $L = 0$.*

Proof. We start by proving that SMM satisfies SP. Under SMM, allocations at round r are independent of allocations at previous rounds. Suppose that agent i reports $d'_{i,r} \neq d_{i,r}$ at round r. Let $a'_{i,r}$ and $a_{i,r}$ denote i's allocations at round r for reporting $d'_{i,r}$ and $d_{i,r}$, respectively. If $a_{i,r} \geq d_{i,r}$, then i already receives her highest possible utility, $d_{i,r}H$ (because $L = 0$), and she cannot benefit from misreporting.

If $a_{i,r} < d_{i,r}$, then for all $j \neq i$, we have: (1) $a_{j,r} \leq d_{j,r}$ and (2) $a_{i,r}/e_i \geq a_{j,r}/e_j$. The former holds by SMM's definition. The latter holds because SMM maximizes the minimum weighted allocations in a lexicographical order. If there is j with $a_{j,r}/e_j > a_{i,r}/e_i$, then SMM should decrease $a_{j,r}$ and increase $a_{i,r}$. Now, suppose for contradiction that $a'_{i,r} > a_{i,r}$. Since $\sum_k a'_{k,r} = \sum_k a_{k,r}$, there should be an agent ℓ with $a'_{\ell,r} < a_{\ell,r} \leq d_{\ell,r}$. Therefore, we have:

$$a'_{\ell,r}/e_\ell < a_{\ell,r}/e_\ell \leq a_{i,r}/e_i < a'_{i,r}/e_i.$$

This is a contradiction because SMM could improve its objective value by decreasing $a'_{i,r}$ and increasing $a'_{\ell,r}$.

To see that SMM satisfies SI, note that an agent can guarantee herself at least e_i resources (her utility from not participating) at each round by reporting $d'_{i,r} = e_i$ for all r. By SP, truthful reporting achieves at least this utility. Therefore, truthful reporting achieves at least as much utility as not participating in SMM, which proves SI. Finally, SMM satisfies efficiency by definition, since it either completely fulfills all demands or allocates all resources to agents that value them highly. □

6.3.2 Properties of Mechanisms for $\mathbf{L > 0}$

We now consider the general setting where an agent's low valuation is still positive. Unfortunately, SMM no longer retains its properties from the $L = 0$ case. Agents are no longer indifferent to forsaking low-valued resources and may lie in order to receive them.

Theorem 8. *When $L > 0$, static max-min mechanism violates strategy-proofness and sharing incentives.*

Proof. Consider an instance with 2 agents, each with endowment $e_i = 1$, and a single round. Agent 1 has demand 2 and agent 2 has demand 0. SMM allocates both resources to agent 1 and nothing to agent 2. However, had agent 2 not participated in the mechanism, she would have received one resource and utility $L > 0$. Similarly, had she misreported her demand to be 1, she would have received one resource and utility $L > 0$. □

Indeed, in this general setting, no mechanism can simultaneously satisfy efficiency and either of the two other desired properties.

Theorem 9. *When $L > 0$, there is no dynamic mechanism that satisfies α-sharing incentives and efficiency, for any $\alpha > 0$.*

Proof. Consider an instance with two agents, each with endowment $e_i = 1$, and a single round. Agent 1 has demand 2 and agent 2 has demand 0. Efficiency dictates that we allocate both resources to agent 1, which would violate α-SI for agent 2 for any $\alpha > 0$. □

Theorem 10. *When $L > 0$, there is no dynamic mechanism that satisfies strategy-proofness and efficiency.*

Proof. Consider an instance with two agents, each with endowment $e_i = 1$, and a single round. Both agents have demand 0. For efficiency, the mechanism must allocate all the resources so that at least one agent receives $a_{i,1} > 0$. Supposing without loss of generality that $a_{1,1} > 0$, then $a_{2,1} < 2$. If agent 2 misreports $d'_{2,1} = 2$, by efficiency, the mechanism must allocate both resources to agent 2, which is an improvement over her utility from reporting truthfully. □

Note that SP and SI are compatible. A mechanism that statically allocates agents their endowments satisfies SP and SI; agents have no incentive to misreport because allocations do not depend on reports and agents receive their fair share of resources. This mechanism clearly fails to satisfy efficiency and does not extract any benefit from sharing. In Section 6.5, we propose a mechanism that satisfies strategy-proofness, guarantees each user at least 50% of their utilities from sharing incentives, and provides an asymptotic efficiency guarantee.

6.4 Proportional Sharing With Constraints Procedure

The mechanisms we present in the remainder of this chapter have, at their core, a procedure we call *Proportional Sharing With Constraints (PSWC)*. The procedure allocates some amount of resources among agents proportionally to their (exogenous) weights subject to (agent-dependent) minimum and limit constraints: (1) each agent receives at least her minimum allocation, and (2) each agent should receives no more than her limit allocation.

Formally, PSWC procedure takes as input an amount to allocate, A, agents' weights, $\mathbf{w} = (w_1, \ldots, w_n)$, agents' minimum allocations, $\mathbf{m} = (m_1, \ldots, m_n)$, and agents' limit allocations, $\mathbf{l} = (l_1, \ldots, l_n)$. PSWC outputs a vector of allocations $\mathbf{a} = (a_1, \ldots, a_n)$ defined as the solution to the following program.

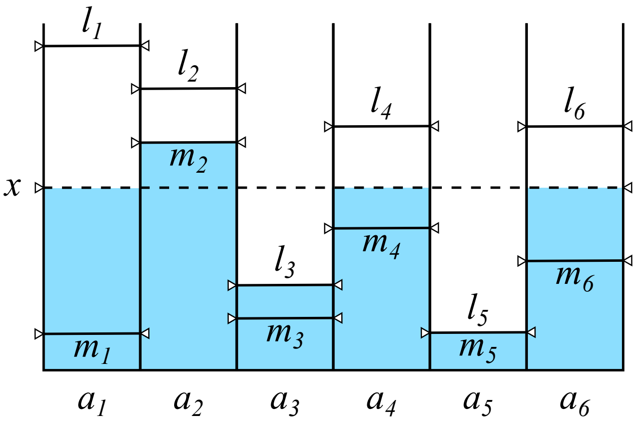


FIGURE 6.2: **Proportional Sharing with Constraints.** For six agents with equal weights, allocations are represented by the height of blue bars.

$$\begin{aligned} &\text{Minimize } x, \\ &\quad \text{s.t. } a_i/w_i \leq x && \text{if } m_i < a_i \leq l_i, \\ &\quad a_i \leq l_i && \forall i, \\ &\quad a_i \geq m_i && \forall i, \\ &\quad \sum_{i \in [n]} a_i = A. \end{aligned}$$

PSWC is illustrated in Figure 6.2. The program can be solved in $O(n \log(n))$ time by the Divvy algorithm [173]. The Divvy algorithm proceeds by sorting the limit and minimum allocation bounds in $O(n \log(n))$ time, and then conducting a linear time search for the optimal value of x by increasing the allocations in discrete steps until all resources have been allocated.

The following lemma characterizes the allocations produced by the PSWC procedure and will be useful in our later proofs.

Lemma 11. *Under PSWC, for every agent i,* $a_i = \max(m_i, \min(l_i, xw_i))$.

Proof. First, we show that if $m_i < xw_i$, then $a_i = \min(l_i, xw_i)$. If $a_i > \min(l_i, xw_i)$, then at least one constraint is violated. If $a_i < \min(l_i, xw_i)$, then there exists at least one agent ℓ such that $a_\ell = xw_\ell$ because otherwise, x is not optimal. In this case, a_i can be increased while a_ℓ for all ℓ with $a_\ell = xw_\ell$ decreases. This allows for a smaller value of x, which contradicts the optimality of x. Next, we show that if $m_i \geq xw_i$, then $a_i = m_i$. Since a_i cannot be less than m_i, if a_i is not equal to m_i, then $a_i > m_i$, which means $a_i > xw_i$. However, since $a_i > m_i$, the first constraint dictates that $a_i \leq xw_i$, a contradiction. Combining these two cases gives the desired result. □

Our proposed mechanisms all have similar structure. First, agents always receive exactly the same number of resources that they contribute to the system (over the entire R rounds). This is a fairness primitive in its own right, but is primarily a design feature that helps us provide desirable properties. Second, all our proposed dynamic mechanisms call the PSWC procedure to allocate resources at each round. Our mechanisms are determined primarily by how we set the minimum and maximum constraints.

6.5 Flexible Lending Mechanism

We now turn to designing mechanisms that satisfy our game-theoretic desiderata while increasing efficiency significantly over static allocation. The static allocation mechanism satisfies both SP and SI, but it does not exhibit any gains from sharing. DMM and SMM sacrifice SP and SI in exchange for efficiency. However, in the absence of SP, any guarantee on efficiency based on agents' demands is not meaningful as agents have incentives to misreport their demands when $L > 0$. In this section, we present the flexible lending (FL) mechanism. The flexible lending mechanism achieves strategy-proofness and an asymptotic efficiency guarantee. FL satisfies a theoretical 0.5 approximation to SI and our simulation results show that it significantly outperforms this bound in practice (see Section 6.7).

6.5.1 Definition

For a fixed number of rounds R, FL allocates exactly Re_i resources to each agent i, which is exactly her contribution to the shared pool over all R rounds. The mechanism enforces this constraint by simply removing agent i from the list of eligible agents once she receives Re_i resources in total. We keep track of the resources each agent has received with a running token count t_i, effectively 'charging' each agent a token for every resource she receives. We denote by $t_{i,r}$ the number of tokens that agent i holds at the start of round r. Thus, the number of tokens that an agent holds puts a hard limit on the number of resources she can receive at any given round.

Algorithm 3 presents the flexible lending mechanism. We define $\bar{d}_i$ to be the allocatable demand of agent i at each round, which is simply the minimum of her reported demand $d'_{i,r}$ and the number of tokens she has remaining t_i. We distinguish between two cases depending on whether the total allocatable demand is higher or lower than the total supply of resources.

First, if the total allocatable demand is at least as high as the total supply, then FL runs PSWC with the minimum allocation for each agent set to 0, and the limit allocation set to $\bar{d}_i$. This way, resources are allocated proportionally among all agents that want them. Second, if the total allocatable demand is less than the total supply, then agents receive their full allocatable demand. Therefore, FL runs PSWC with minimum allocation for each agent i set to $\bar{d}_i$, and limit allocations set to her number of tokens t_i (which is always at least as large as her allocatable demand). This way, FL allocates resources proportionally among all agents, subject to the condition that no agent receives fewer resources than her demand.

We illustrate FL with an example.

Example 12. *Consider a system with three agents and four rounds. Each agent has endowment $e_i = 1$. Suppose that agents' (truthful) reports are given by the following*

Algorithm 3: Flexible Lending Mechanism

```
$\mathbf{t} = R\mathbf{e}$                                          ▷ Initialize token count
for $r \in \{1, \dots, R\}$ do
    $\bar{\mathbf{d}} \leftarrow \min(\mathbf{d'_{\cdot,r}}, \mathbf{t})$                     ▷ $\bar{d}_i$ is $i$'s allocatable demand
    $D \leftarrow \sum_{i \in [n]} \bar{d}_i$
    if $D \geq E$ then
        $\mathbf{a_{\cdot,r}} \leftarrow \text{PSWC}(A = E, \mathbf{l} = \bar{\mathbf{d}}, \mathbf{m} = \mathbf{0}, \mathbf{w} = \mathbf{e})$
    end
    else
        $\mathbf{a_{\cdot,r}} \leftarrow \text{PSWC}(A = E, \mathbf{l} = \mathbf{t}, \mathbf{m} = \bar{\mathbf{d}}, \mathbf{w} = \mathbf{e})$
    end
    $\mathbf{t} \leftarrow \mathbf{t} - \mathbf{a_{\cdot,r}}$
end
```

table:

	$d_{i,1}$	$d_{i,2}$	$d_{i,3}$	$d_{i,4}$
$i = 1$	*3*	*1*	*1*	*0*
$i = 2$	*0*	*2*	*1*	*2*
$i = 3$	*0*	*0*	*0*	*4*

FL allocations are given by the following table:

	$a^{FL}_{i,1}$	$a^{FL}_{i,2}$	$a^{FL}_{i,3}$	$a^{FL}_{i,4}$
$i = 1$	*3*	*1*	*0*	*0*
$i = 2$	*0*	*2*	*1.5*	*0.5*
$i = 3$	*0*	*0*	*1.5*	*2.5*

While all agents have tokens remaining, FL efficiently allocates resources. However, in round 3, agent 1 has no tokens remaining and therefore the supply of resources exceeds the allocatable demand. In this case, resources are evenly divided between agents 2 and 3. In the final round, agent 2 can receive only 0.5 resources before running out of tokens, so the rest of the resources are allocated to agent 3.

6.5.2 Basic Properties

Next, we study the properties of FL. We first show that FL satisfies strategy-proofness. We then show that FL guarantees at least 50% of SI performance. And finally we show that FL provides an asymptotic efficiency guarantee. Throughout this section, we extensively use the following lemma which characterizes FL allocations.

Lemma 13. *Let x denote the objective value of FL's call to PSWC at round r. If $D \geq E$, then $a_{i,r} = \min(xe_i, d_{i,r}, t_{i,r})$. If $D < E$, then $a_{i,r} = \min(t_{i,r}, \max(d_{i,r}, xe_i))$.*

Proof. Suppose first that $D \geq E$. Substituting the relevant terms into Lemma 11, we have

$$a_{i,r} = \max(0, \min(\min(d_{i,r}, t_{i,r}), xe_i)) = \min(xe_i, d_{i,r}, t_{i,r}).$$

If instead $D < E$, then again substituting into Lemma 11 gives

$$a_{i,r} = \max(\min(d_{i,r}, t_{i,r}), \min(t_{i,r}, xe_i)) = \min(t_{i,r}, \max(d_{i,r}, xe_i)).$$

The final equality, $\max(\min(A, B), \min(A, C)) = \min(A, \max(B, C))$ can easily be checked to hold case by case for any relative ordering of A, B, and C. □

We next prove a basic monotonicity result, which states that if we shift some tokens to a single agent from all other agents, then the agent with more tokens achieves a (weakly) higher allocation. The proof follows easily from Lemma 13 and is deferred to the Appendix.

Lemma 14. *Consider some agent i, and suppose that $t'_{i,r} \geq t_{i,r}$, $t'_{j,r} \leq t_{j,r}$ for all $j \neq i$, and $d'_{k,r} = d_{k,r}$ for all $k \in [n]$. Then $a'_{i,r} \geq a_{i,r}$.*

As our main technical result, we show in the following subsection that FL is strategy-proof. At a high level, we show that if an agent receives fewer high-valued resources as a result of misreporting, then her allocations in all future rounds are weakly higher. This means that she cannot receive fewer low-valued resources at any future round, relative to her allocations had she not misreported. Therefore, because the total number of resources allocated to each agent is fixed (by the initial token count), her misreport can only result in trading high-valued resources at an early round for other, potentially low-valued, resources at later rounds.

6.5.3 Strategy-Proofness

Suppose agent i reports demands that are not equal to her true demands. Let r' be the latest round for which i misreports. That is, $r' = \max\{r : d'_{i,r} \neq d_{i,r}\}$. Suppose that $d'_{i,r'} < d_{i,r'}$. We show that, all else being equal, i could (weakly) improve her utility by instead reporting $d'_{i,r'} = d_{i,r'}$. The proof that reporting $d'_{i,r'} > d_{i,r'}$ is also (weakly) worse than reporting $d'_{i,r'} = d_{i,r'}$ is almost identical and can be found in Appendix A.12. It follows from this that FL is strategy-proof, since any non-truthful reports can be converted to truthful reports one round at a time, (weakly) improving i's utility.

We consider parallel universes: one in which agent i misreports $d'_{i,r'}$ at round r' (the 'misreported instance') and one in which she truthfully reports $d_{i,r}$ (the 'truthful instance,' even though i's reports prior to r' may yet be non-truthful). All other reports are identical in both universes. We denote allocations and tokens in the misreported instance using a' and t', respectively, and in the truthful instance by a and t. We denote by D_r and D'_r the total demand D at round r in the truthful and misreported instances, respectively.

We first note that for all rounds prior to r', the allocations in the truthful and misreported instances are the same.

Lemma 15. *For all rounds $r < r'$ and for all agents j, $a'_{j,r} = a_{j,r}$.*

Proof. The mechanism does not take future reports into account, so because agents' demands in both instances are identical up to round r', so are the allocations. □

We next show a monotonicity lemma, which says that agent i's allocation at round r' is (weakly) smaller in the misreported instance than the truthful instance, and all other agents' allocations are (weakly) larger.

Lemma 16. *For all agents $j \neq i$, we have that $a'_{j,r'} \geq a_{j,r'}$. Further, $a'_{i,r'} \leq a_{i,r'}$.*

Proof. We prove the statement for all $j \neq i$. The statement for i follows immediately because the total number of allocated resources is fixed. Observe first that

$$D'_{r'} = \sum_{k \in [n]} \min(d'_{k,r'}, t_{k,r'}) \leq \sum_{k \in [n]} \min(d_{k,r'}, t_{k,r'}) = D_{r'},$$

since i's demand decreases in the misreported instances but all other demands and token counts stay the same. Let x' denote the objective value in FL's call to PSWC in the misreported instance, and x in the truthful instance.

Suppose that $E \leq D'_{r'} \leq D_{r'}$. Suppose first that $x' > x$. Then, by Lemma 13, for all $j \neq i$, we have

$$a'_{j,r'} = \min(x' e_j, d_{j,r'}, t_{j,r'}) \geq \min(x e_j, d_{j,r'}, t_{j,r'}) = a_{j,r'}.$$

Next, suppose that $x' \leq x$. Then, again by Lemma 13 and the fact that $d'_{i,r'} < d_{i,r'}$,

$$a'_{i,r'} = \min(x' e_i, d'_{i,r'}, t_{i,r'}) \leq \min(x e_i, d_{i,r'}, t_{i,r'}) = a_{i,r'}.$$

And, for all $j \neq i$,

$$a'_{j,r'} = \min(x' e_j, d_{j,r'}, t_{j,r'}) \leq \min(x e_j, d_{j,r'}, t_{j,r'}) = a_{j,r'}.$$

Because $a'_{k,r'} \leq a_{k,r'}$ for all agents k, and $\sum_{k \in [n]} a_{k,r'} = \sum_{k \in [n]} a'_{k,r'}$, it must be the case that $a'_{k,r'} = a_{k,r'}$ for all k, which satisfies the statement of the lemma.

Next, suppose that $D'_{r'} < E \leq D_{r'}$. By the definition of FL, $a'_{k,r'} \geq \min(d'_{k,r'}, t_{k,r'})$ for all k, and $a_{k,r'} \leq \min(d_{k,r'}, t_{k,r'})$ for all k. Since $\min(d'_{j,r'}, t_{j,r'}) = \min(d_{j,r'}, t_{j,r'})$ for all $j \neq i$, we have that $a'_{j,r'} \geq a_{j,r'}$, implying also that $a'_{i,r'} \leq a_{i,r'}$.

Finally, suppose that $D'_{r'} \leq D_{r'} < E$. Suppose first that $x' \leq x$. Then, by Lemma 13 and the assumption that $d'_{i,r'} < d_{i,r'}$, we have

$$a'_{i,r'} = \min(t_{i,r'}, \max(x' e_i, d'_{i,r'})) \leq \min(t_{i,r'}, \max(x e_i, d_{i,r'})) = a_{i,r'}$$

and

$$a'_{j,r'} = \min(t_{j,r'}, \max(x'e_j, d_{j,r'})) \leq \min(t_{j,r'}, \max(xe_j, d_{j,r'})) = a_{j,r'}$$

for all $j \neq i$. Because $a'_{k,r'} \leq a_{k,r'}$ for all agents k, and $\sum_{k \in [n]} a_{k,r'} = \sum_{k \in [n]} a'_{k,r'}$, it must be the case that $a'_{k,r'} = a_{k,r'}$ for all k, which satisfies the lemma's statement. Next, suppose that $x' > x$. Then, again by Lemma 13, for all $j \neq i$, we have

$$a'_{j,r'} = \min(t_{j,r'}, \max(x'e_j, d_{j,r'})) \geq \min(t_{j,r'}, \max(xe_j, d_{j,r'})) = a_{j,r'}.$$

□

If it is the case that $a'_{i,r'} = a_{i,r'}$, then it must also be the case that $a'_{j,r'} = a_{j,r'}$ for all $j \neq i$. That is, allocations at round r' are the same in the misreported instance as the truthful instance. Therefore, for all rounds $r \leq r'$, allocations in both universes would be the same. In all rounds $r > r'$, reports in both universes are the same. Together, these imply that allocations for all rounds $r > r'$ would be the same in both universes. In particular, i does not profit from her misreport and could weakly improve her utility by reporting $d'_{i,r'} = d_{i,r'}$. So, for the remainder of this section, we assume that $a'_{i,r'} < a_{i,r'}$.

Our next lemma states that the resources that i sacrifices in round r' are *high*-valued resources for her. The intuition is that if it were the case that i was being forced to receive low-valued resources under truthful reporting, then she will still be forced to receive the same number of resources when she under-reports her demand (since there is no agent with excess demand to absorb extra resources).

Lemma 17. *If $a'_{i,r'} < a_{i,r'}$, then $a_{i,r'} \leq d_{i,r'}$.*

Proof. Suppose for contradiction that $a_{i,r'} > d_{i,r'}$. It must therefore be the case that $D'_{r'} \leq D_{r'} < E$, where the first inequality holds because $d'_{j,r'} = d_{j,r'}$ for all $j \neq i$ and $d'_{i,r'} < d_{i,r'}$. Let x denote the objective value of FL's call to PSWC in the truthful instance, and x' in the misreported instance. Suppose that $x' \leq x$. Then, by Lemma 13 and the assumption that $d'_{i,r'} < d_{i,r'}$,

$$a'_{i,r'} = \min(t_{i,r'}, \max(x'e_i, d'_{i,r'})) \leq \min(t_{i,r'}, \max(xe_i, d_{i,r'})) = a_{i,r'},$$

and for all $j \neq i$,

$$a'_{j,r'} = \min(t_{j,r'}, \max(x'e_j, d_{j,r'})) \leq \min(t_{j,r'}, \max(xe_j, d_{j,r'})) = a_{j,r'}.$$

Because $a'_{k,r'} \leq a_{k,r'}$ for all agents k, and $\sum_{k\in[n]} a_{k,r'} = \sum_{k\in[n]} a'_{k,r'}$, it must be the case that $a'_{k,r'} = a_{k,r'}$ for all k. This contradicts the assumption that $a'_{i,r'} < a_{i,r'}$.

Now suppose that $x' > x$. Note that $xe_i > d_{i,r'} > d'_{i,r'}$, where the first inequality holds because $a_{i,r'} > d_{i,r'}$. Then, again by Lemma 13 and the previous observation, we have

$$\begin{aligned} a'_{i,r'} &= \min(t_{i,r'}, \max(x'e_i, d'_{i,r'})) = \min(t_{i,r'}, x'e_i) \\ &\geq \min(t_{i,r'}, xe_i) = \min(t_{i,r'}, \max(xe_i, d_{i,r'})) = a_{i,r'}, \end{aligned}$$

which contradicts $a'_{i,r} < a_{i,r}$. Since we arrive at a contradiction in all cases, the lemma statement must be true. □

As a corollary, we can write the difference in utility between the truthful and misreported instances that i derives from round r'.

Corollary 18. $u_{i,r'}(a_{i,r'}) - u_{i,r'}(a'_{i,r'}) = H(a_{i,r'} - a'_{i,r'})$.

Proof. Because $a'_{i,r'} < a_{i,r'} \leq d_{i,r'}$, we can substitute the utility values from Equation (6.2):

$$u_{i,r'}(a_{i,r'}) - u_{i,r'}(a'_{i,r'}) = a_{i,r'}H - a'_{i,r'}H = H(a_{i,r'} - a'_{i,r'}).$$

□

For a fixed agent k, denote by r_k the round at which agent k runs out of tokens in the truthful instance. That is, r_k is the first (and only) round with $a_{r_k} = t_{k,r_k} > 0$.

Note that $r_i \geq r'$, since $a_{i,r'} > 0$. Given this, our next lemma states that, under certain conditions, the effect of i's misreport, $d'_{i,r} < d_{i,r}$, is to increase the objective value of FL's call to PSWC.

Lemma 19. *Let $r < r_i$ (i.e. $a_{i,r} < t_{i,r}$). Suppose $t'_{j,r} \leq t_{j,r}$ for all agents $j \neq i$. Suppose that either $\min(D_r, D'_r) \geq E$ or $\max(D_r, D'_r) < E$. Then $x' \geq x$, where x' denotes the objective value of FL's call to PSWC in the misreported instance and x in the truthful instance.*

Proof. First, suppose that $\min(D_r, D'_r) \geq E$. Suppose for contradiction that $x' < x$. By Lemma 13, for all $j \neq i$,

$$a'_{j,r} = \min(x'e_j, d_{j,r}, t'_{j,r}) \leq \min(xe_j, d_{j,r}, t_{j,r}) = a_{j,r},$$

where the inequality follows from the assumption that $x' < x$ and that $t'_{j,r} \leq t_{j,r}$. Further,

$$a'_{i,r} = \min(x'e_i, d_{i,r}, t'_{i,r}) \leq \min(x'e_i, d_{i,r}) \leq \min(xe_i, d_{i,r}) = \min(xe_i, d_{i,r}, t_{i,r}) = a_{i,r},$$

where the second inequality follows from the assumption that $x' < x$, and the second to the last equality follows from the assumption that $a_{i,r} < t_{i,r}$. Therefore, $a'_{k,r} \leq a_{k,r}$ for all agents k. Since $\sum a'_{k,r} = \sum a_{k,r}$, it must be the case that $a'_{k,r} = a_{k,r}$ for all agents k. Now, by the definition of FL in this case, $a_{k,r}/e_k \leq x' < x$ for all agents k with $a_{k,r} > 0$. Therefore x is not the optimal objective value of PSWC in the truthful instance, a contradiction. Thus, $x' \geq x$.

Next, suppose that $\max(D_r, D'_r) < E$. Suppose for contradiction that $x' < x$. By Lemma 13,

$$a'_{j,r} = \min(t'_{j,r}, \max(x'e_j, d_{j,r})) \leq \min(t_{j,r}, \max(xe_j, d_{j,r})) = a_{j,r},$$

for all $j \neq i$, where the inequality follows from the assumption that $x' < x$ and that

$t'_{j,r} \leq t_{j,r}$. Further, we have

$$\begin{aligned} a'_{i,r} &= \min(t'_{i,r}, \max(x'e_i, d_{i,r})) \leq \max(x'e_i, d_{i,r}) \\ &\leq \max(xe_i, d_{i,r}) = \min(t_{i,r}, \max(xe_i, d_{i,r})) = a_{i,r}, \end{aligned}$$

where the second inequality follows from the assumption that $x' < x$ and the second to last equality from the assumption $a_{i,r} < t_{i,r}$. Therefore, $a'_{k,r} \leq a_{k,r}$ for all agents k. Since $\sum a'_{k,r} = \sum a_{k,r}$, it must be the case that $a'_{k,r} = a_{k,r}$ for all agents k. Consider all agents with $\min(d_{k,r}, t_{k,r}) < a_{k,r}$ (*i.e.* those agents for which the first constraint in the PSWC program binds in the truthful instance). For all such agents, we have

$$\begin{aligned} \min(d_{k,r}, t_{k,r}) < a_{k,r} &\implies d_{k,r} < a_{k,r} \leq t_{k,r} \implies d_{k,r} < a'_{k,r} \leq t'_{k,r} \\ &\implies \min(d_{k,r}, t'_{k,r}) < a'_{k,r}, \end{aligned}$$

which implies that the constraints bind in the misreported instance as well. Therefore, $a'_{k,r}/e_k \leq x' < x$ for all agents k for which the first constraint binds in the truthful instance. Therefore x is not the optimal objective value of the PSWC program in the truthful instance, a contradiction. Thus, $x' \geq x$. □

Using Lemma 19, we show our main lemma. This lemma allows us to make an inductive argument that, after giving up some resources in round r', i's allocation is (weakly) larger for all future rounds in the misreported instance than the truthful instance.

Lemma 20. *Let $r' < r < r_i$ (i.e. $a_{i,r} < t_{i,r}$). Suppose that $t'_{j,r} \leq t_{j,r}$ for all agents $j \neq i$. Then for all $j \neq i$, either: (1) $a'_{j,r} = t'_{j,r}$, or (2) $a'_{j,r} \geq a_{j,r}$.*

Proof. Note that $t'_{j,r} \leq t_{j,r}$ for all $j \neq i$ implies that $t'_{i,r} \geq t_{i,r}$, which we use in the proof. Also, because $r > r'$, we know that $d'_{i,r} = d_{i,r}$, as r' is the last round for which $d'_{i,r} \neq d_{i,r}$. We assume that condition (1) from the lemma statement is false (*i.e.* $a'_{j,r} < t'_{j,r}$) and show that condition (2) must hold. Suppose first that $D_r < E$. Then, because $a_{i,r} < t_{i,r}$, we know that $d_{i,r} \leq t_{i,r} \leq t'_{i,r}$. This implies that $\min(d_{i,r}, t_{i,r}) = \min(d_{i,r}, t'_{i,r}) = d_{i,r}$. Let $j \neq i$. Since $t'_{j,r} \leq t_{j,r}$, we have $\min(d_{j,r}, t'_{j,r}) \leq \min(d_{j,r}, t_{j,r})$. Therefore, it is the case that $D'_r \leq D_r < E$. By Lemma 13 and the assumption that $a'_{j,r} < t'_{j,r}$, it must be the case that $a'_{j,r} = \max(d_{j,r}, x'e_j)$. Further, by Lemma 19, we know that $x' \geq x$. Therefore, we have

$$a_{j,r} = \max(d_{j,r}, xe_j) \leq \max(d_{j,r}, x'e_j) = a'_{j,r}.$$

That is, condition (2) from the lemma statement holds.

Now suppose that $D_r \geq E$. Then, from the definition of the mechanism, we have that $a_{j,r} \leq \min(d_{j,r}, t_{j,r}) \leq d_{j,r}$. If it is the case that $D'_r < E$, then we have that $a'_{j,r} \geq \min(d_{j,r}, t'_{j,r}) = d_{j,r}$, where the equality holds because otherwise we would have $a'_{j,r} \geq t'_{j,r}$, violating the assumption that $a'_{j,r} < t'_{j,r}$. Using these inequalities, we have $a'_{j,r} \geq d_{j,r} \geq a_{j,r}$, so condition (2) from the statement of the lemma holds. Finally, it may be the case that $D_r \geq E$ and $D'_r \geq E$. By Lemma 13 and the assumption that $a'_{j,r} < t'_{j,r}$, we have

$$a'_{j,r} = \min(d_{j,r}, x'e_k) \geq \min(d_{j,r}, xe_k) = a_{j,r},$$

where the inequality follows from Lemma 19. Thus, condition (2) of the lemma statement holds. □

Finally, we prove that the flexible lending mechanism is strategy-proof. This proof establishes that misreporting $d'_{i,r}$ is never beneficial for an agent.

Theorem 21. *The flexible lending mechanism satisfies SP.*

Proof. We first observe that for every $r \leq r_i$, $t'_{j,r} \leq t_{j,r}$ for every $j \neq i$. This is true for every $r \leq r'$ because $a'_{j,r} = a_{j,r}$ for $r < r'$, by Lemma 15. For $r = r' + 1$, it follows from Lemma 16, which says that $a'_{j,r'} \geq a_{j,r'}$. For all subsequent rounds, up to and including $r = r_i$, it follows inductively from Lemma 20: $t'_{j,r} \leq t_{j,r}$ implies that either $a'_{j,r} = t'_{j,r}$, in which case $t'_{j,r+1} = 0 \leq t_{j,r+1}$, or $a'_{j,r} \geq a_{j,r}$, in which case $t'_{j,r+1} = t'_{j,r} - a'_{j,r} \leq t_{j,r} - a_{j,r} = t_{j,r+1}$).

Consider an arbitrary round $r \neq r'$, with $r \leq r_i$. By the above argument, we know that $t'_{j,r} \leq t_{j,r}$ for all $j \neq i$. Further, because reports in the truthful and misreported instances are identical on all rounds $r \neq r'$, we have that $d_{k,r} = d'_{k,r}$ for all $k \in [n]$. Therefore, by Lemma 14, $a'_{i,r} \geq a_{i,r}$. For rounds $r > r_i$, it is also true that $a'_{i,r} \geq a_{i,r}$, since $a_{i,r} = 0$ for these rounds by the definition of r_i. Finally,

$$\begin{aligned}
U_{i,R}(\mathbf{a_i}) - U_{i,R}(\mathbf{a'_i}) &= \sum_{r=1}^{R}(u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r})) \\
&= (u_{i,r'}(a_{i,r'}) - u_{i,r'}(a'_{i,r'})) + \sum_{r \neq r'}(u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r})) \\
&= H(a_{i,r'} - a'_{i,r'}) - \sum_{r \neq r'}(u_{i,r}(a'_{i,r}) - u_{i,r}(a_{i,r})) \\
&\geq H(a_{i,r'} - a'_{i,r'}) - H(a_{i,r'} - a'_{i,r'}) = 0
\end{aligned}$$

Here, the third transition follows from Lemma 18, and the final transition follows because $\sum_{r \neq r'}(a'_{i,r} - a_{i,r}) = a_{i,r'} - a'_{i,r'}$, and every term in the sum is positive.

The proof for the case where $d'_{i,r'} > d_{i,r'}$ is in the Appendix. Together, they show that i achieves (weakly) higher utility by truthfully reporting her demand $d_{i,r'}$ at round r, rather than misreporting $d'_{i,r'} \neq d_{i,r'}$. By the argument at the start of this subsection, this is sufficient to prove strategy-proofness. □

6.5.4 Approximating Sharing Incentives

Unfortunately, FL fails to satisfy SI, and may give an agent as little as half of her SI share.

Theorem 22. *FL does not satisfy α-SI for any $\alpha > 0.5$.*

Proof. Consider an instance with R rounds, and $R+1$ agents, each with endowment $e_i = 1$. Agent 1 has $d_{1,1} = d_{1,R} = 1$ and $d_{1,2} = \ldots = d_{1,R-1} = 0$, agent 2 has

$d_{2,r} = R$ for all rounds r, and all other agents have $d_{i,r} = 0$ for all rounds r. In round 1, agent 1 receives allocation $a_{1,1} = 1$ and agent 2 receives $a_{2,1} = R$. For rounds $r = 2, \ldots, R-1$, each agent $j \neq 2$ receives allocation $a_{j,r} = 1 + 1/R$. Therefore, in round R, agent 1 receives $a_{1,R} = R-1-(R-2)(1+1/R) = 2/R$. Her total utility is therefore $((R+2)/R)H + (R-(R+2)/R)L$, compared to total utility $2H+(R-2)L$ that she would have received by not participating in the mechanism. For $L = 0$, the ratio of these utilities approaches 0.5 as $R \to \infty$. □

However, FL does provide a 0.5 approximation guarantee to SI, as we show in the remainder of this subsection. We suppose that agent i truthfully reports her demand $d_{i,r}$ for all rounds (since FL is SP, she could do no better by lying), and show that she receives at least half as much utility as she would by not participating.

Recall that for every agent i, we denote by r_i the first round at which $a_{i,r_i} = t_{i,r_i} > 0$. For every agent i, define sets B_i and A_i to be the agents that run out of tokens before and after i, respectively. Formally,

$$B_i = \{j : r_j \leq r_i \text{ and } a_{j,r_i}/e_j < a_{i,r_i}/e_i\}$$

$$A_i = \{j : r_j \geq r_i \text{ and } r_j = r_i \implies a_{j,r_i}/e_j \geq a_{i,r_i}/e_i\}.$$

For a round r, define

$$s_{i,r} = a_{i,r} - e_i \frac{\sum_{j\in A_i} a_{j,r}}{\sum_{j\in A_i} e_j}.$$

That is, $s_{i,r}$ is the number of resources i gets more than the (endowment weighted) average number of resources for agents in A_i. Note further that

$$\sum_{r=1}^{R} s_{i,r} = \sum_{r=1}^{R} a_{i,r} - \frac{e_i}{\sum_{j\in A_i} e_j} \sum_{j\in A_i} \sum_{r=1}^{R} a_{j,r} = e_i - \frac{e_i}{\sum_{j\in A_i} e_j} \sum_{j\in A_i} e_j = 0.$$

Lemma 23. *For every agent i and every round r, $s_{i,r} \leq \min(d_{i,r}, a_{i,r})$.*

Proof. If $a_{i,r} \leq d_{i,r}$, then the lemma statement says that $s_{i,r} \leq a_{i,r}$, which is obviously true by the definition of $s_{i,r}$. If $a_{i,r} > d_{i,r}$, then we know from the definition of FL that $\sum_{j \in [n]} \min(d_{j,r}, t_{j,r}) < E$, and $a_{i,r} = \min(xe_i, t_{i,r})$, where x is the objective value of FL's call to the PSWC program. Further, all agents with $\frac{a_{j,r}}{e_j} < \frac{a_{i,r}}{e_i} \leq x$ are those with $a_{j,r} = t_{j,r}$, so by definition, $r_j \leq r_i$ and $\frac{a_{j,r}}{e_j} < \frac{a_{i,r}}{e_i}$, which means $j \in B_i$. Therefore, $\frac{a_{j,r}}{e_j} \geq \frac{a_{i,r}}{e_i}$ for all $j \in A_i$, which implies $\frac{\sum_{j \in A_i} a_{j,r}}{\sum_{j \in A_i} e_j} \geq \frac{a_{i,r}}{e_i}$. To complete the proof, note that

$$s_{i,r} = a_{i,r} - e_i \frac{\sum_{j \in A_i} a_{j,r}}{\sum_{j \in A_i} e_j} \leq a_{i,r} - e_i \frac{a_{i,r}}{e_i} = 0 \leq d_{i,r} = \min(d_{i,r}, a_{i,r}).$$

□

Theorem 24. *Under FL, agents receive at least half the number of high-valued resources that they would have received under static allocations.*

Proof. Let S denote the number of high-valued resources that agent i receives under static allocations. While i has tokens remaining, under FL, she is guaranteed to get as many resources as she demands up to her endowment e_i. Thus, for these rounds, she would obtain no additional high-valued resources from not participating in the mechanism. However, there is the possibility that by participating in the mechanism, she runs out of tokens prematurely, thus missing out on resources in later rounds that she wants, and would have received by not participating in the mechanism (as in the proof of Theorem 22). The proof proceeds by showing that for every resource that i does not receive due to a lack of tokens, she must have received at least one high-valued resource in an earlier round.

Suppose first that $a_{i,r_i} \geq e_i$. We have the following inequality:

$$\sum_{r \leq r_i} \min(d_{i,r}, a_{i,r}) \geq \sum_{r \leq r_i} s_{i,r} = -\sum_{r > r_i} s_{i,r} = \sum_{r > r_i} \left(e_i \frac{\sum_{j \in A_i} a_{j,r}}{\sum_{j \in A_i} e_j} \right) \tag{6.1}$$

$$= \sum_{r > r_i} \left(\frac{E}{\sum_{j \in A_i} e_j} \right) e_i \geq (T - r_i) e_i. \tag{6.2}$$

The first inequality follows from Lemma 23, and the second inequity follows because $\sum_{j \in A_i} e_j \leq E$. The first equality holds because $\sum_{r=1}^{R} s_{i,r} = 0$, and the second equality holds because $a_{i,r} = 0$ for all $r > r_i$. The third equality holds because for rounds $r > r_i$, only agents in A_i remain active, so all resources are allocated to them.

Note that S, the number of high-valued resources that i receives by not sharing, is upper bounded by

$$\begin{aligned} S \leq \sum_{r=1}^{R} \min(d_{i,r}, e_i) &\leq \sum_{r \leq r_i} \min(d_{i,r}, e_i) + \sum_{r > r_i} e_i \\ &\leq \sum_{r \leq r_i} \min(d_{i,r}, a_{i,r}) + \sum_{r > r_i} e_i \\ &= \sum_{r \leq r_i} \min(d_{i,r}, a_{i,r}) + (T - r_i) e_i \\ &\leq 2 \sum_{r \leq r_i} \min(d_{i,r}, a_{i,r}). \end{aligned}$$

The third inequality holds because under FL guarantees each agent $\min(d_{i,r}, e_i)$ resources, provided they have sufficient tokens remaining, which is the case because we assume $a_{i,r_i} \geq e_i$. The final inequality follows from Equation (6.1). Since agent i receives exactly $\sum_{r \leq r_i} \min(d_{i,r}, a_{i,r}) \geq S/2$ resources from participating in FL, the lemma holds in this case.

Second, suppose that $a_{i,r_i} < e_i$. We have the following inequality:

$$\begin{aligned}
\sum_{r \le r_i} \min(d_{i,r}, a_{i,r}) &\ge \sum_{r < r_i} \min(d_{i,r}, a_{i,r}) \\
&\ge \sum_{r < r_i} s_{i,r} = -\sum_{r > r_i} s_{i,r} - s_{i,r_i} \\
&\ge e_i(T - r_i) + e_i \frac{\sum_{j \in A_i} a_{j,r_i}}{\sum_{j \in A_j} e_j} - a_{i,r_i} \\
&\ge e_i(T - r_i) + e_i - a_{i,r_i} = e_i(T - r_i + 1) - a_{i,r_i}. \qquad (6.3)
\end{aligned}$$

The first inequality holds because $\min(d_{i,r_i}, a_{i,r_i}) \ge 0$. The second inequality follows from Lemma 23, and the third inequality holds from Equation (6.1) and the definition of s_{i,r_i}. The fourth inequality holds because at round r_i, agent i receives allocation $a_{i,r_i} < e_i$, therefore every agent $j \in B_i$ receives allocation $a_{j,r_i} < e_j$, therefore $\sum_{j \in A_i} a_{j,r_i} \ge \sum_{j \in A_i} e_j$.

As with the previous case, we can derive an upper bound on S, the number of high-valued resources i would receive by not sharing. First, suppose that $a_{i,r_i} > d_{i,r_i}$. Then we have

$$\begin{aligned}
S \le \sum_{r=1}^{R} \min(d_{i,r}, e_i) &\le \sum_{r < r_i} \min(d_{i,r}, e_i) + d_{i,r_i} + \sum_{r > r_i} e_i \\
&\le \sum_{r < r_i} \min(d_{i,r}, a_{i,r}) + \min(d_{i,r_i}, a_{i,r_i}) + \sum_{r > r_i} e_i \\
&= \sum_{r \le r_i} \min(d_{i,r}, a_{i,r}) + (T - r_i)e_i \\
&\le \sum_{r \le r_i} \min(d_{i,r}, a_{i,r}) + (T - r_i + 1)e_i - a_{i,r_i} \\
&\le 2\sum_{r \le r_i} \min(d_{i,r}, a_{i,r})
\end{aligned}$$

The third inequality holds because FL guarantees each agent $\min(d_{i,r}, e_i)$ resources, provided they have sufficient tokens remaining, and by the assumption that $a_{i,r_i} >$

d_{i,r_i}, the fourth inequality from the assumption that $a_{i,r_i} < e_i$, and the final inequality from Equation (6.3). Next, suppose that $a_{i,r_i} \leq d_{i,r_i}$. Then we have

$$\begin{aligned} S \leq \sum_{r=1}^{R} \min(d_{i,r}, e_i) &\leq \sum_{r<r_i} \min(d_{i,r}, e_i) + e_i + \sum_{r>r_i} e_i \\ &\leq \sum_{r<r_i} \min(d_{i,r}, a_{i,r}) + a_{i,r_i} + (e_i - a_{i,r_i}) + \sum_{r>r_i} e_i \\ &= \sum_{r\leq r_i} \min(d_{i,r}, a_{i,r}) + (T - r_i + 1)e_i - a_{i,r_i} \\ &\leq 2\sum_{r\leq r_i} \min(d_{i,r}, a_{i,r}) \end{aligned}$$

The third inequality holds because FL guarantees each agent $\min(d_{i,r}, e_i)$ resources, provided they have sufficient tokens remaining, the equality from the assumption that $a_{i,r_i} \leq d_{i,r_i}$, and the final inequality from Equation (6.3).

As with the previous case, $e_i(T - r_i + 1) - a_{i,r_i}$ is an upper bound on the number of H valued resources that i may have been able to receive in rounds $r \geq r_i$ had she not participated in the mechanism, over and above those she receives by participating. $\sum_{r\leq r_i} \min(d_{i,r}, a_{i,r})$ is the number of H valued resources she receives by participating in the mechanism. Therefore $\sum_{r\leq r_i} \min(d_{i,r}, a_{i,r}) + e_i(T - r_i + 1) - a_{i,r_i} \leq 2\sum_{r\leq r_i} \min(d_{i,r}, a_{i,r})$ is an upper bound on the number of H valued resources i would receive by not participating in the mechanism. Therefore, i receives at least half as many H valued resources from participating as she would have by not participating. □

Note that Theorem 24 implies the desired approximation. Suppose that i obtains utility $SH + (Re_i - S)L$ by not participating in the mechanism. Theorem 24 in combination with the fact that she will receive the same number of resources overall whether she participates or not, implies that, by participating, she gets at least $SH/2 + (Re_i - S/2)L \geq SH/2 + (Re_i/2 - S/2)L = (SH + (Re_i - S)L)/2$.

6.5.5 Limit Efficiency for Symmetric Agents

In this section, we prove that, under certain assumptions, FL is efficient in the limit as the number of rounds grows large. Suppose that each agent has the same endowment. Without loss of generality, suppose that each agent has $e_i = 1$. Further, suppose that demands are drawn i.i.d. across rounds and that the distribution within rounds treats agents symmetrically, either demands are drawn i.i.d. across agents, or there is correlation that treats all agents symmetrically.

Theorem 25. *When demands are drawn i.i.d. across rounds and agents are symmetric, FL achieves an $(R - R^{2/3})/R$ fraction of the optimal efficiency with probability at least $1 - n^3/R^{1/3}$. In particular, FL approaches full efficiency with high probability in the limit as the number of rounds grows large.*

Proof. Suppose we are in a world where tokens are unlimited. Let Q be a random variable denoting how many tokens a single agent i would spend (*i.e.* how many resources i would be allocated) in a single round. Note that Q can never take a value larger than n, since only n resources are allocated per round. Note that by the symmetry of the agents, Q is independent of the identity of any single agent, and independent of the particular round since FL allocates independently of the round. By symmetry, $\mathbb{E}(Q) = 1$. Let $\text{StdDev}(Q) = \sigma \leq n$, where the inequality holds because Q is bounded by n. Let $r = R - R^{2/3}$ and let Q_r be a random variable denoting the number of tokens i would spend before the start of round $r+1$. Because demands are drawn independently across rounds, and no agent runs out of tokens, $\mathbb{E}(Q_r) = r$ and $\text{StdDev}(Q_r) = \sqrt{r}\sigma$.

Consider the probability that agent i spends at least R tokens in the first r rounds:

$$\begin{aligned}
P(Q_r \geq R) &= P(Q_r - \mathbb{E}(Q_r) \geq R - r) \\
&= P(Q_r - \mathbb{E}(Q_r) \geq R^{2/3} \\
&= P(Q_r - \mathbb{E}(Q_r) \geq \frac{R^{1/6}}{\sigma}\sqrt{R}\sigma) \\
&\leq P(Q_r - \mathbb{E}(Q_r) \geq \frac{R^{1/6}}{\sigma}\sqrt{r}\sigma) \\
&\leq \frac{\sigma^2}{R^{1/3}}
\end{aligned}$$

Here the final inequality follows from Chebyshev's concentration inequality, because $\sqrt{r}\sigma$ is the standard deviation of Q_r. Taking a union bound over all n agents, the probability that any agent spends at least R tokens in the first r rounds is at most $n\sigma^2/R^{1/3} \leq n^3/R^{1/3}$.

Now suppose agents are limited by R tokens. If some agent runs out of tokens within r rounds in this world, then it must also be the case that some agent spent at least R tokens within r rounds in the unlimited token world. Therefore, the probability that any agent runs out of tokens is at most the probability that some agent spends more than R tokens in the unlimited token world, which is at most $n^3/R^{1/3}$. This approaches 0 as $R \to \infty$. So, with probability going to 1, no agent runs out of tokens before round r.

By the definition of FL, full efficiency is achieved on all rounds for which no agents have their allocation limited by lack of tokens. Therefore, with probability going to 1, FL allocates efficiently for the first r rounds. Therefore, because demands are i.i.d. across rounds, the expected efficiency of the mechanism approaches at least an $r/R = (R - R^{2/3})/R$ fraction of the optimal efficiency. This fraction approaches 1 as $R \to \infty$. □

6.6 **T**-Period Mechanism

We have shown that FL satisfies strategy-proofness and a theoretical asymptotic efficiency guarantee. Further, as we show in Section 6.7, FL exhibits only small efficiency loss in practice in settings where our theoretical guarantee does not apply. However, FL does not achieve (full) sharing incentives. In settings where agents require a strong guarantee in order to participate, it may be desirable to strictly enforce sharing incentives, in which case FL is not a suitable choice. In this section, we introduce the *T-Period mechanism*, which satisfies both SP and SI. While the T-Period mechanism does exhibit some gains from sharing (i.e., is more efficient than static allocation), it sacrifices some efficiency relative to FL.

6.6.1 Definition

The T-Period mechanism splits the rounds into periods of length $2T$.[1] For the first T rounds of each period, we allow the agents to 'borrow' unwanted resources from others. In the last T rounds of each period, the agents 'pay back' the resources so that their cumulative allocation across the entire period is equal to their endowment, $2Te_i$.

The allocations in the second set of T rounds are independent of reports and determined completely by the allocations in the first set of T rounds. Note that because the number of resources that an agent i can pay back over T rounds is bounded by Te_i, we allow an agent to borrow at most Te_i resources (i.e., receive at most $2Te_i$ resources) over the first T rounds of a period.

In Algorithm 4, each agent i has a borrowing limit, b_i, which is defined to be the maximum amount of resources that agent i can borrow in whatever remains of the first T rounds of each period. For our analysis, we denote the value of b_i at the start

[1] For convenience, we suppose that R is a multiple of $2T$. If this is not the case, we can adapt the mechanism by returning each agent their endowment for any leftover rounds.

Algorithm 4: T-Period Mechanism

```
input  : Agents' reported demands, $\mathbf{d'}$, and their endowments, $\mathbf{e}$
output: Agents' allocations, $\mathbf{a}$
for $r \in \{1, \ldots, R\}$ do
    if $(r \ mod \ 2T) = 1$ then
        $\mathbf{b} \leftarrow T\mathbf{e}$                    ▷ $b_i$ is the amount that $i$ is able to borrow
        $\mathbf{y} \leftarrow \mathbf{0}$                       ▷ resources received so far this period.
    end
    if $1 \leq (r \ mod \ 2T) \leq T$ then
        $\bar{\mathbf{d}} \leftarrow \min(\mathbf{d'_{\cdot,r}}, \mathbf{e} + \mathbf{b})$          ▷ $\bar{d}_i$ is $i$'s allocatable demand
        $D \leftarrow \sum_{i \in [n]} \bar{d}_i$
        if $D \geq E$ then
            $\mathbf{a_{\cdot,r}} \leftarrow \text{PSWC}(A = E, \mathbf{l} = \bar{\mathbf{d}}, \mathbf{m} = \mathbf{0}, \mathbf{w} = \mathbf{e})$
        end
        else
            $\mathbf{a_{\cdot,r}} \leftarrow \text{PSWC}(A = E, \mathbf{l} = \mathbf{e} + \mathbf{b}, \mathbf{m} = \bar{\mathbf{d}}, \mathbf{w} = \mathbf{e})$
        end
        $\mathbf{y} \leftarrow \mathbf{y} + \mathbf{a_{\cdot,r}}$
        $\mathbf{b} \leftarrow \mathbf{b} - \max(\mathbf{0}, \mathbf{a_{\cdot,r}} - \mathbf{e})$
    end
    else
        $\mathbf{a_{\cdot,r}} \leftarrow \frac{1}{T}(2T\mathbf{e} - \mathbf{y})$
    end
end
```

of round r by $b_{i,r}$. At the beginning of each period, we set $b_{i,r}$ to be Te_i, because agent i can at most pay back her whole endowment, e_i, at every T 'payback' rounds. We again define $\bar{d}_i$ to be the allocatable demand of agent i at each round of the first T rounds and refer to $\bar{d}'_{i,r}$ as agent i's allocatable demand at round r. At each round r, the allocatable demand of agent i is the minimum of her reported demand $d'_{i,r}$, and her endowment plus her borrowing limit, $e_i + b_{i,r}$.

We illustrate the T-Period mechanism with an example.

Example 26. *Consider the instance from Example 12, where each agent has endowment $e_i = 1$ and demands are given by:*

	$d_{i,1}$	$d_{i,2}$	$d_{i,3}$	$d_{i,4}$
$i = 1$	*3*	*1*	*1*	*0*
$i = 2$	*0*	*2*	*1*	*2*
$i = 3$	*0*	*0*	*0*	*4*

When $T = 1$*, agents can 'borrow' resources at odd rounds and 'pay back' those resources at even rounds. Therefore, the maximum allocatable demand for each agent and at each round is 2, because the 'payback' period only has one round. The* 1-Period (1-P) *mechanism allocates resources as follows.*

	$a_{i,1}^{1\text{-}P}$	$a_{i,2}^{1\text{-}P}$	$a_{i,3}^{1\text{-}P}$	$a_{i,4}^{1\text{-}P}$
$i = 1$	*2*	*0*	*1*	*1*
$i = 2$	*0.5*	*1.5*	*1*	*1*
$i = 3$	*0.5*	*1.5*	*1*	*1*

At round 1, agent 1 wants 2 extra resources in addition to her endowment. However, under 1-P, she can only afford 1 extra resource. She borrows 0.5 resources from agent 2 and 0.5 resources from agent 3. At round 2, agent 1 pays back agents 2 and 3 and receives zero resources. When $T = 1$*, the mechanism rigidly forces agents to pay back resources right after they borrow them. Agent 1 would prefer to get her high-valued resource at round 2 and delay paying back agents 2 and 3 until the last round where her demand is zero. Note that agent 3 would also prefer to be paid back in the last round, the only round in which she has non-zero demand.*

To see how increasing T allows more flexibility, consider $T = 2$ *for the same example. The* 2-Period (2-P) *mechanism allocates resources as follows.*

	$a_{i,1}^{2\text{-}P}$	$a_{i,2}^{2\text{-}P}$	$a_{i,3}^{2\text{-}P}$	$a_{i,4}^{2\text{-}P}$
$i = 1$	*3*	*1*	*0*	*0*
$i = 2$	*0*	*2*	*1*	*1*
$i = 3$	*0*	*0*	*2*	*2*

Agent 2 is allowed to borrow 2 extra resources over the first two rounds, whereas, under the 1-P mechanism, she is never allowed to borrow more than one resource per round. She borrows these two resources at the first round from agents 2 and 3, and *pays them back at rounds 3 and 4.*

Since the T-Period mechanism increases flexibility over the static mechanism, it provides some gains from sharing. We would expect that increasing T, in general, will improve efficiency as it allows for 'borrowed' resources to be spent more flexibly. In the following subsection, we show that these efficiency gains do not harm SI or SP when $T \leq 2$. Many proofs closely follow those in Section 6.5 and are deferred to the Appendix.

6.6.2 Axiomatic Properties of **T***-Period Mechanism*

We first state a lemma characterizing the allocations of the T-Period mechanism that is analogous to Lemma 13

Lemma 27. *Let x denote the objective value of a call to PSWC. Suppose that $1 \leq (r \mod 2T) \leq T$. If $D \geq E$, then $a_{i,r} = \min(e_i + b_i, d'_{i,r}, xe_i)$. If $D < E$, then $a_{i,r} = \min(e_i + b_i, \max(d'_{i,r}, xe_i))$.*

To prove strategy-proofness of the 1-Period and 2-Period mechanisms, we show that no agent has an incentive to report $d'_{i,r} \neq d_{i,r}$ for any round r. We again consider parallel cases, one in which agent i misreports $d'_{i,r}$ and one in which she truthfully reports $d_{i,r}$ with all other reports the same across the two cases. Allocations and borrowing limits in the former case is denoted by a' and b' respectively, and by a and b in the latter case. Let D_r denote the total allocatable demand at a round r in the truthful case and D'_r denote the total allocatable demand at a round r in the misreported case.

Since the T-Period mechanism resets every $2T$ rounds, we can assume without loss of generality that $R = 2T$ for the sake of reasoning about SP and SI. For rounds $r > T$, the allocations depend completely on the allocations at earlier rounds, and not on the agents' reports, so there is clearly no benefit to an agent for misreporting in these rounds. It remains to show that reporting $d'_{i,r} = d_{i,r}$ is optimal for rounds $r \leq T$.

Our next lemma is analogous to Lemma 16.

Lemma 28. *Let $a_{i,r}$ and $a'_{i,r}$ denote the allocations of agent i at round r when she reports $d_{i,r}$ and $d'_{i,r}$, respectively, holding fixed the reports of all agents $j \neq i$ and agent i's reports on all rounds other than r. If $d'_{i,r} < d_{i,r}$ then $a'_{i,r} \leq a_{i,r}$, and $a'_{j,r} \geq a_{j,r}$ for all $j \neq i$.*

Suppose that i reports $d'_{i,r} \neq d_{i,r}$ for some round r, but this misreport does not change i's allocation (that is, $a'_{i,r} = a_{i,r}$). By Lemma 28, $a'_{j,r} = a_{j,r}$ for all $j \neq i$. Therefore, i's misreport has not changed the allocations at round r. Since all future rounds take into account allocations at previous rounds but not reports, i's misreport has had no effect on the allocations in any round. Thus, i did not benefit from this misreport. We therefore assume that $a'_{i,r} \neq a_{i,r}$ for any round r where i reports $d'_{i,r} \neq d_{i,r}$ in the remainder of this section.

The next lemma and corollary are analogous to Lemma 17 and Corollary 18. They say that if i obtains fewer resources from misreporting at round r, then those resources are all high-valued resources.

Lemma 29. *Hold the reports of all agents $j \neq i$ fixed, and the reports of agent i on all rounds other than r fixed. If i reports $d'_{i,r} < d_{i,r}$ and receives $a'_{i,r} < a_{i,r}$, then $a_{i,r} \leq d_{i,r}$.*

As a corollary we obtain a formula for the difference between the utility that agent i receives at round r under truthful reporting and misreporting, when i gets fewer resources in the misreported instance.

Corollary 30. *Hold the reports of all agents $j \neq i$ fixed, and the reports of agent i on all rounds other than r fixed. If i reports $d'_{i,r} < d_{i,r}$ and receives $a'_{i,r} < a_{i,r}$, then $u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r}) = H(a_{i,r} - a'_{i,r})$.*

The next lemma and corollary complement Lemma 29 and Corollary 30 in the case where i receives more resources in the misreported instance than the truthful instance at round r.

Lemma 31. *Hold the reports of all agents $j \neq i$ fixed, and the reports of agent i on all rounds other than r fixed. If i reports $d'_{i,r} > d_{i,r}$ and receives $a'_{i,r} > a_{i,r}$, then $a_{i,r} \geq d_{i,r}$.*

Corollary 32. *Hold the reports of all agents $j \neq i$ fixed, and the reports of agent i on all rounds other than r fixed. If i reports $d'_{i,r} > d_{i,r}$ and receives $a'_{i,r} > a_{i,r}$, then $u_{i,r}(a'_{i,r}) - u_{i,r}(a_{i,r}) = L(a'_{i,r} - a_{i,r})$.*

We can now show that misreporting in round T is never beneficial to an agent.

Lemma 33. *An agent never improves her utility by reporting $d'_{i,T} \neq d_{i,T}$.*

As a corollary, we immediately have that the 1-Period mechanism is strategy-proof, because misreporting at round $r = 1 = T$ is not beneficial, and misreporting at round $r = 2 > T$ is not beneficial by our earlier argument.

Corollary 34. *The 1-Period mechanism satisfies strategy-proofness.*

Our next lemma is a monotonicity statement for the borrowing limits: if i's borrowing limit at round r increases, and all other agents' borrowing limits decrease, then i's allocation (weakly) increases and all other agents' allocations (weakly) decrease.

Lemma 35. *Suppose that $r \leq T$. If $b'_{i,r} \geq b_{i,r}$ and $b'_{j,r} \leq b_{j,r}$ for all $j \neq i$, and $d'_{k,r} = d_{k,r}$ for all agents k, then $a'_{i,r} \geq a_{i,r}$.*

We now show that the 2-Period mechanism is strategy-proof.

Theorem 36. *The 2-Period mechanism satisfies strategy-proofness.*

Proof. By Lemma 33, no agent can benefit by reporting $d'_{i,2} \neq d_{i,2}$. Similarly, no agent can benefit by reporting $d'_{i,r} \neq d_{i,r}$ for $r \in \{3, 4\}$, because the 2-Period mechanism ignores reports for those rounds. We may therefore assume that $d'_{i,r} = d_{i,r}$ for all agents i and all rounds $r \geq 2$.

We show that an agent cannot benefit from reporting $d'_{i,1} < d_{i,1}$. The proof that reporting $d'_{i,1} > d_{i,1}$ is not beneficial is very similar. If $a'_{i,1} = a_{i,1}$, then $a'_{j,1} = a_{j,1}$ for all $j \neq i$, by Lemma 28. Therefore, the allocations are unchanged for all rounds i, as the 2-Period mechanism takes into account allocations at earlier rounds, but not reports, and the allocations at round 1 are the same in the truthful and misreported instances. We therefore assume that $a_{i,1} = a'_{i,1} + k$, for some $k > 0$. This implies that $b_{i,2} = b'_{i,2} - k_i$, for some $k_i \leq k$. By Corollary 30, i receives kH more utility in round 1 under truthful reporting than under misreporting. For every $j \neq i$, $a_{j,1} \leq a'_{j,1}$, and $b_{j,2} = b'_{j,2} + k_j$, where $\sum_{j \neq i} k_j \leq k$. By Lemma 35, $a'_{i,2} \geq a_{i,2}$. In the following, we show that $a'_{i,2} \leq a_{i,2} + k$. Let x and x' denote the objective value in the T-Period mechanism's call to PSWC when i reports $d_{i,r}$ and $d'_{i,r}$, respectively. We consider four cases, corresponding to whether resources in the truthful and misreported instances are over or under demanded at round 2. Suppose first that $D_2 \geq E$ and $D'_2 \geq E$. First, suppose that $x' < x$. Then, by Lemma 27,

$$\begin{aligned} a'_{i,2} &= \min(e_i + b'_i, d_{i,2}, x'e_i) = \min(e_i + b_{i,2} + k_i, d_{i,2}, x'e_i) \\ &\leq \min(e_i + b_{i,2}, d_{i,2}, x'e_i) + k_i \leq \min(e_i + b_{i,2}, d_{i,2}, xe_i) + k_i \leq a_{i,2} + k \end{aligned}$$

Next, suppose that $x' \geq x$. Then for all $j \neq i$,

$$\begin{aligned} a'_{j,2} &= \min(e_j + b'_j, d_{j,2}, x'e_j) = \min(e_j + b_{j,2} - k_j, d_{j,2}, x'e_j) \\ &\geq \min(e_j + b_{j,2}, d_{j,2}, x'e_j) - k_j \geq \min(e_j + b_{j,2}, d_{j,2}, xe_j) - k_j = a_{j,2} - k_j \end{aligned}$$

Taking the sum over all $j \neq i$ and noting that $\sum_{j \neq i} k_j \leq k$, we have that $\sum_{j \neq i} a'_{j,2} \geq \sum_{j \neq i} a_{j,2} - k$. Therefore, $a'_{i,2} \leq a_{i,2} + k$. Second, suppose that $D_2 \geq E$ and $D'_2 < E$.

Then, by the definition of the T-Period mechanism, $a_{j,2} \leq \min(e_j + b_{j,2}, d_{j,2})$ for all $j \neq i$. Further

$$\begin{aligned} a'_{j,2} \geq \min(e_j + b'_j, d_{j,2}) &= \min(e_j + b_{j,2} - k_j, d_{j,2}) \\ &\geq \min(e_j + b_{j,2}, d_{j,2}) - k_j \geq a_{j,2} - k_j \end{aligned}$$

By the same argument as in the previous case, this implies that $a'_{i,2} \leq a_{i,2} + k$. Third, suppose that $D_2 < E$ and $D'_2 \geq E$. Then

$$a'_{i,2} \leq \min(e_i + b'_i, d_{i,2}) = \min(e_i + b_{i,2} + k_i, d_{i,2}) \leq \min(e_i + b_{i,2}, d_{i,2}) + k_i \leq a_{i,r} + k$$

Finally, suppose that $D_2 < E$ and $D'_2 < E$. First, suppose that $x' < x$. Then

$$\begin{aligned} a'_{i,2} = \min(e_i + b'_i, \max(d_{i,2}, x'e_i)) &= \min(e_i + b_{i,2} + k_i, \max(d_{i,2}, x'e_i)) \\ &\leq \min(e_i + b_{i,2}, \max(d_{i,2}, x'e_i)) + k_i \\ &\leq \min(e_i + b_{i,2}, \max(d_{i,2}, xe_i)) + k_i \leq a_{i,2} + k \end{aligned}$$

Next, suppose that $x' \geq x$. Then for all $j \neq i$,

$$\begin{aligned} a'_{j,2} = \min(e_j + b'_j, \max(d_{j,2}, x'e_j)) &= \min(e_j + b_{j,2} - k_j, \max(d_{j,2}, x'e_j)) \\ &\geq \min(e_j + b_{j,2}, \max(d_{j,2}, x'e_j)) - k_j \\ &\geq \min(e_j + b_{j,2}, \max(d_{j,2}, xe_j)) - k_j = a_{j,2} - k_j \end{aligned}$$

Again, this implies that $a'_{i,2} \leq a_{i,2} + k$.

In all cases, we have that $a_{i,2} \leq a'_{i,2} \leq a_{i,2} + k$. Therefore, $a'_{i,1} + a'_{i,2} \leq a_{i,1} + a_{i,2}$, which means that $a'_{i,3} \geq a_{i,3}$ and $a'_{i,4} \geq a_{i,4}$. Consider the difference in utility across all four rounds between the truthful and misreported instances.

$$\begin{aligned} U_{i,4}(\mathbf{a_i}) - U_{i,4}(\mathbf{a'_i}) &= \sum_{r=1}^{4} \left(u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r})\right) \\ &= kH + \sum_{r=2}^{4} \left(u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r})\right) \geq kH - kH = 0 \end{aligned}$$

The second transition is by Corollary 30, and the third transition because each $a'_{i,r} \geq a_{i,r}$ for all $r \in \{2,3,4\}$, $\sum_{r=2}^{4}(a'_{i,r} - a_{i,r}) = k$, and each resource can be worth at most H to agent i. □

Given that the 1-P and 2-P mechanisms satisfy SP, it is easy to see that they satisfy SI also. By strategy-proofness, the utility that an agent gets from truthfully reporting her demands is at least the utility she gets from reporting $d'_{i,r} = e_i$ for all rounds r. Sharing incentives therefore follows as a corollary of the following proposition.

Proposition 37. *Under the T-Period mechanism, any agent that reports $d'_{i,r} = e_i$ for all rounds r receives $a_{i,r} = e_i$ for all rounds r.*

Corollary 38. *The T-Period mechanism satisfies SI for $T \leq 2$.*

One may hope to continue increasing flexibility, and therefore performance, by increasing the length of the 'borrowing' and 'payback' periods, potentially all the way to having a single borrowing period of length $R/2$ and a single payback period of length $R/2$. Unfortunately, even for periods of length 3, strategy-proofness is violated.

Example 39. *Consider the 3-P mechanism. Suppose that $n = 5$ and $R = 6$. Each agent has endowment $e_i = 1$ (so each agent can borrow a total of three resources over the first three rounds, corresponding to the sum of their endowment across the final three rounds). Truthful demands are given by the following table.*

	$d_{i,1}$	$d_{i,2}$	$d_{i,3}$	$d_{i,4}$	$d_{i,5}$	$d_{i,6}$
$i = 1$	*3*	*3*	*0*	*1*	*1*	*1*
$i = 2$	*0*	*3*	*3*	*1*	*1*	*1*
$i = 3$	*0*	*0*	*0*	*0*	*0*	*0*
$i = 4$	*0*	*0*	*0*	*0*	*0*	*0*
$i = 5$	*0*	*0*	*0*	*0*	*0*	*0*

The corresponding allocations are given by:

	$a^{3-P}_{i,1}$	$a^{3-P}_{i,2}$	$a^{3-P}_{i,3}$	$a^{3-P}_{i,4}$	$a^{3-P}_{i,5}$	$a^{3-P}_{i,6}$
$i = 1$	*3*	*2*	*0.75*	*0.08*	*0.08*	*0.08*
$i = 2$	*0.5*	*3*	*2*	*0.17*	*0.17*	*0.17*
$i = 3$	*0.5*	*0*	*0.75*	*1.58*	*1.58*	*1.58*
$i = 4$	*0.5*	*0*	*0.75*	*1.58*	*1.58*	*1.58*
$i = 5$	*0.5*	*0*	*0.75*	*1.58*	*1.58*	*1.58*

Agent 1's utility is $5.25H + 0.75L$. *If agent 1 misreports* $d'_{1,1} = 2$, *it can be checked that her allocations become 2, 2.5, 0.625, 0.292, 0.292, 0.292. Her utility is then* $5.375H + 0.625L$, *which is higher than her utility from reporting truthfully.*

6.7 Evaluation

In this section, we evaluate different mechanisms using real and synthetic benchmarks. For real benchmarks, we use a Google cluster trace [174, 175], which data collected from a 12.5k-machine cluster over a month-long period in May 2011. All the machines in the cluster share a common cluster manager that allocates agent tasks to machines.

Agents submit a set of resource demands for each task (*e.g.* required processors, memory, or disk space). Agent demands are normalized relative to the largest capacity of the resource on any machine in the traces. The cluster manager records any changes in the status of tasks (*e.g.* being evicted, failed, or killed) during their life cycle in a task event table. We use the task event table to track agents' demands for processors over time. Note that since all demands are scaled by the same factor, we safely use normalized demands as actual demands.

We divide time into 15 min intervals.[2] We define agents' demands for each interval to be the sum of their demands for all tasks they run in that interval. After processing the traces, we remove agents with constant demands or with average demand less

[2] We have created demands for varying time intervals. Since results do not change significantly for different interval lengths, we only include results on 15-min-long intervals.

than some marginal threshold. We assume that agents' endowments are equal to their average demands.

We observe that, for each agent, demands computed from Google traces have high correlations over time. An agent with high demand at 12am has typically high demand at 12:15am as well. In some deployment scenarios, demands may not be highly correlated. For example, when university cluster machines are allocated to professors and researchers on a daily basis, a researcher may have some jobs today, but may not want to use the cluster tomorrow.

To evaluate mechanisms in scenarios without correlated demands, we use synthetic benchmarks. We create random agent populations and random number of rounds. For each agent, we uniformly and randomly assign an endowment from 1 to 20. Once agents' endowments are set, we uniformly and randomly generate agent demands such that their average is equal to agents' endowments (*i.e.* $d_{i,r} \sim u[0, 2e_i]$)

Metrics. We report social welfare and Nash welfare, focusing on the number of high-valued resources that each mechanism allocates. For social welfare, we report the following.

$$\text{Social Welfare} = \sum_i \sum_r \min(d_{i,r}, a_{i,r}).$$

Social welfare is a measure of efficiency but fails to distinguish between fair and unfair outcomes. For instance, suppose agent A with endowment 100 and agent B with endowment 1 both have demand 101. Allocating 100 units to agent A and 1 unit to agent B has the same social welfare as allocating 1 unit to A and 100 units to B. To distinguish between these two allocations, we also report the (weighted) Nash welfare as follows.

$$\text{Nash Welfare} = \sum_i e_i \log(\sum_r \min(d_{i,r}, a_{i,r})).$$

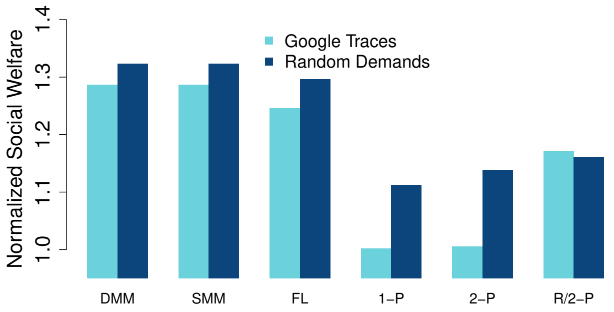


FIGURE 6.3: **Normalized Social Welfare.** Social welfare achieved by different dynamic allocation mechanisms normalized to that of static allocations for Google cluster traces and 100 instances of random demands.

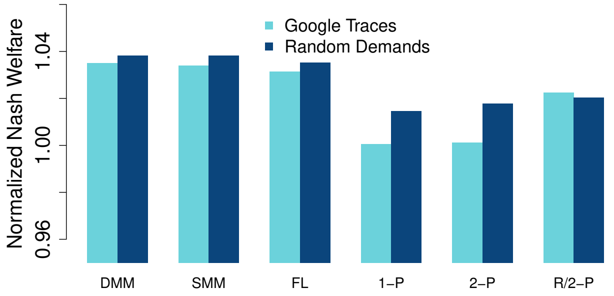


FIGURE 6.4: **Normalized Nash Welfare.** The Nash welfare achieved by different dynamic allocation mechanisms normalized to that of static allocations for Google cluster traces and 100 instances of random demands.

Observe that the Nash welfare metric is higher for the former scenario than the latter, which is in line with our intuition about which allocation is more fair.

6.7.1 Performance Evaluation

Figure 6.3 presents social welfare from varied allocation mechanisms for both Google and random traces normalized to social welfare of static allocations. DMM and SMM produce the same, highest social welfare as they always allocate resources to those agents with high valuations. Note that SMM and DMM both fail to guarantee strategy-proofness when $L > 0$. Therefore, when agents report strategically, for all the mechanism knows, SMM and DMM's allocations could be as inefficient as static

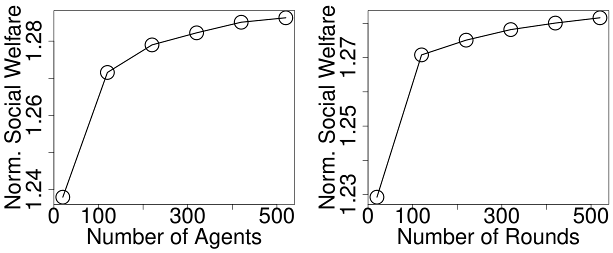


FIGURE 6.5: **Sensitivity of the Flexible Lending Mechanism.** Social welfare of the flexible lending mechanism normalized to that of static allocations for varying agent population sizes and numbers of rounds.

allocations. But this is not captured in the figure, which implicitly assumes truthful reporting.

The 1-Period mechanism produces the lowest social welfare. Increasing the period length to 2 slightly improves the welfare of the T-Period mechanism. Note that both mechanisms outperform static allocations. The $R/2$-Period mechanism achieves 87% of SMM welfare for Google traces, but fails to provide strategy-proofness.

The social welfare of FL is competitive with state-of-the-art dynamic allocation mechanisms. FL achieves 97% of SMM's welfare for Google traces and 98% for random demands. In practice, strong game-theoretic desiderata do not come with high welfare costs.

Figure 6.4 compares the normalized Nash welfare from varied mechanisms. Once again, DMM and SMM outperform other mechanisms, but DMM and SMM's outcomes are no longer equal because the number of high-valued resources that each agent receives differs across mechanisms. FL achieves 99.7% of DMM welfare for both Google cluster and random traces. This high Nash welfare could be explained by FL's high social welfare and the fact that FL allocates agents their exact endowment across rounds.

Figure 6.5 shows how social welfare changes when varying population size and number of rounds under FL. As the population size increases, the diversity between

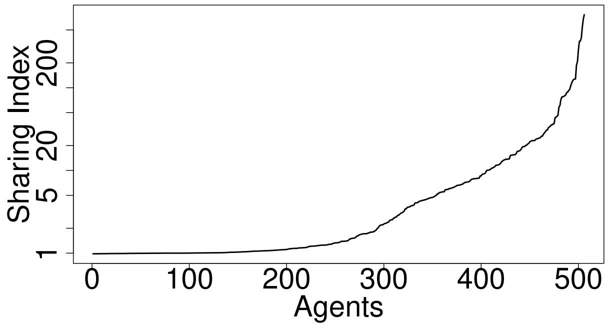


FIGURE 6.6: **Sorted Sharing Index for Google Cluster Traces.**

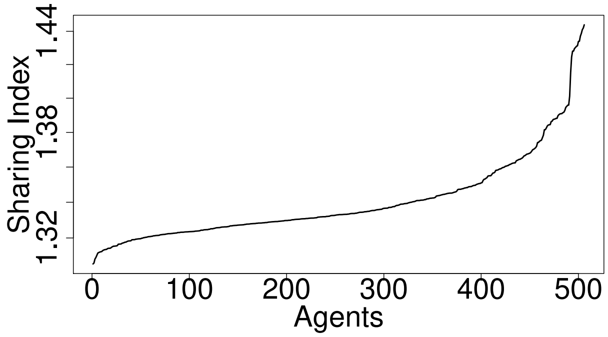


FIGURE 6.7: **Sorted Sharing Index for a Random Trace.**

agents' demands at each round increases. Agents' complementary demands improve welfare from FL as fewer agents are forced to spend tokens on low-valued resources. Moreover, as the number of rounds increases, agents' flexibility in spending their tokens on high-valued resources increases. We prove in Section 6.5.5 that, at least when endowments are equal, FL approximates efficiency.

6.7.2 Sharing Incentives

We define the *sharing index* of agent i to be the ratio between the number of high-valued resources agent i receives under FL and under static allocations. In Section 6.5.4, we show that FL guarantees that the sharing index of each agent is always at least 0.5. In practice, however, our simulations show that the sharing index is much higher.

Figure 6.6 shows the sharing index for all agents in the Google cluster traces, sorted in increasing order and shown on a log scale. The minimum sharing index across all agents is 0.98, and on average agents receive 15x more utility under FL compared to static allocations. As can be seen, there is high variance in sharing index across agents. Agents with high index are those who have zero demand at most of the rounds and very high demand at a few rounds. These agents benefit the most from sharing. When they have zero demand, they do not spend any tokens. Once they have a high demand they spend their tokens to receive the resources they need.

Figure 6.7 shows agents' sharing index for an instance with random demands. Since agents do not have correlated demands, the variance in sharing index is significantly lower compared to the Google cluster traces. Moreover, across all agents over 100 random instances, we do not observe a single violation of SI (*i.e.* no agent has a sharing index less than 1)

6.8 Related Work

There is a body of work in the *mechanism design without money* literature that is related to our work. Gorokh et al. [110] consider a setting where a single item is to be allocated repeatedly, and extend to more general settings in a follow-up paper [176]. They do so by endowing each user with a fixed amount of artificial currency and then treating it similarly to if it were real money. They show that, for a large enough number of rounds, incentives to misreport and welfare loss both vanish. However, their notion of strategy-proofness is ex-ante Bayesian, requiring users (and the mechanism) to know the distribution from which other users' demands are drawn and truthful reporting is optimal only in expectation. Our notion of SP is ex-post, meaning that an agent never regrets truthful reporting.

Various other work does not explicitly use artificial currency, but by keeping track of how much utility an agent should receive in the future, achieve guarantees in a way that resembles the use of artificial currency [177, 178, 179]. Again, these results are for a weaker notion of SP.

In a similar setting, Aleksandrov et al. [172, 180] consider a stream of resources arriving one at a time that must be allocated among competing strategic agents. They consider two mechanisms, one of which is similar to SMM and the other similar to DMM. They obtain both positive and negative results for these mechanisms, however their positive results are primarily obtained for the case where agent utilities are 0 or 1, corresponding to our $L = 0$ case. They also consider only the symmetric agent setting, rather than our setting that allows unequal endowments.

There also exists literature on *dynamic fair division* [181, 182, 183], but this work predominantly focuses on agents arriving and departing over time, rather than the preferences themselves being dynamic, as in our work.

In the systems literature, in recent years, there has been a growing body of work on using economic game-theory to allocate resources [8, 1, 108]. These works only consider one-shot allocations and do not study allocations over time. Ghodsi et al. [111] consider dynamic allocations over time but in a completely different allocation setting than ours. Their proposed mechanism allocates resources to packets in a queue. In such a setting, time cannot be divided into fixed intervals, because processing packets take different times, which means a packet could stall all other packets until its processed. As a result, proportional allocations have to be approximated through discrete packet scheduling decisions [112, 31].

To allocate CPU time proportionally across applications, Duda et al. [184] propose Borrowed-Virtual-Time (BVT) Scheduling and show that it provides low-latency for real-time and interactive applications for general-purpose systems. In BVT, applications effectively borrow virtual time from their future CPU allocation and thus do not disrupt long-term CPU sharing. Although this idea is very close to the main idea of the flexible lending mechanism, unlike FL, BVT does not provide any guarantees on strategy-proofness.

In a work that is close to our setting, Tang et al. [185] propose a dynamic allocation policy that resembles DMM. We study the characteristics of DMM in Section 6.3 and evaluate its performance in Section 6.7. Another related work in this area is that of [186]. The authors propose a scheduler that allocates resources between users with dynamically changing demands. This work deploys heuristics and does not provide any theoretical guarantees that we study in this chapter.

6.9 Conclusion

We have considered the problem of designing mechanisms for dynamic proportional sharing in a high-low utility model that both incentivize users to participate and share their resources (sharing incentives), as well as truthfully report their resource requirements to the system (strategy-proofness). We show that while each of these properties is incompatible with full efficiency, it is possible to satisfy both of them and still obtain some efficiency gains from sharing.

The main mechanism that we present, the flexible lending mechanism, is strategy-proof and provides each user a theoretical guarantee of at least half her sharing incentives share. While we do not guarantee full sharing incentives, we show via simulations on both real and synthetic data that in practical situations, no users are significantly worse off by participating in the sharing scheme (and the majority are vastly better off). We show that under certain assumptions, the flexible lending mechanism provides full efficiency in the large round limit, which is supported by our simulation results. By incentivizing truthful reporting, we posit that the flexible lending mechanism will in fact produce significant efficiency gains in settings where agents are strategic.

Many directions for future work remain. The 2-Period mechanism fully satisfies both SP and SI, but remains very inflexible in its allocations. A key challenge is the design of a more flexible mechanism that satisfies both properties (or some upper bound on the efficiency that such mechanisms can achieve). Another direction is to extend the utility model. The high/low model is crucial to the positive strategic results that we obtain because trade-offs are well-defined: swapping an L resource for an H resource is always bad. Even introducing a medium (M) value complicates the situation considerably, and extending to such a setting would represent an exciting step forward.

7

Conclusion

Drawing on game theory, this thesis encourages new thinking in designing management platforms robust to strategic behavior. In this thesis, we present five pieces of work on data center management platforms. First, REF provides an allocation mechanism to encourage sharing. We show that Cobb-Douglas utilities are well suited to modeling user preferences in computer architecture. For Cobb-Douglas utilities, we present an allocation mechanism that provides sharing incentives, envy-freeness, Pareto efficiency, and strategy-proofness in the large. By linking hardware resource management to robust, game-theoretic analysis, computer architects can qualitatively change the nature of performance guarantees in hardware platforms shared by strategic users.

Second, computation sprinting game presents a management framework that determines when each chip should sprint. We formalize sprint management as a repeated game. Agents represent chip multiprocessors and their workloads, executing sprints strategically on their behalf. Strategic behaviors produce an equilibrium in the game. We show that, in equilibrium, the computational sprinting game outperforms prior, greedy mechanisms by 4-6× and delivers 90% of the performance achieved from a more expensive, globally enforced mechanism.

Third, our proposed token mechanism presents a new approach for fair resource management in dynamic systems. The token system provides game-theoretic desiderata while offering flexibility, which enhances performance by allocating resources to jobs in time periods that benefit performance most. We demonstrate a fair, repeated allocation game for heterogeneous processors that generalizes to other resources.

Fourth, the Amdahl utility function concisely models user value from processor core allocations. We present a profiling framework that operationalizes Amdahl's Law, using its inverse—the Karp-Flat metric—to estimate the parallelizable fraction of a workload. We co-design with the Amdahl utility function, a market mechanism and a novel bidding procedure to allocate processors. Allocations ensure entitlements in a shared datacenter.

Finally, the flexible lending mechanism dynamically allocate resources between users with high-low utility model. We prove that our proposed mechanism is strategy-proof and provides each user a theoretical guarantee of at least half her sharing incentives share. While we do not guarantee full sharing incentives, we show via simulations on both real and synthetic data that in practical situations, no users are significantly worse off by participating in the sharing scheme (and the majority are vastly better off). We show that under certain assumptions, the flexible lending mechanism provides full efficiency in the large round limit, which is supported by our simulation results. By incentivizing truthful reporting, we posit that the flexible lending mechanism will in fact produce significant efficiency gains in settings where agents are strategic.

Appendix A

Omitted Proofs

A.1 Proof of Strategy Proofness in the Large For REF

Suppose user i decides to lie about her utility function, reporting α'_{ir} instead of the true value α_{ir} for resource r. Given other users' utilities, user i would choose to report the α'_{ir} that maximizes her utility. As mentioned in Section 2.4, in a large system $1 \ll \sum_j \alpha_{jr}$, for all resources r. Since $\alpha'_{ir} \leq 1$, we have $\alpha'_{ir} \ll \sum_j \alpha_{jr}$. Therefore:

$$u_i = \prod_{r=1}^{R} \left(\frac{\alpha'_{ir}}{\alpha'_{ir} + \sum_{j \neq i} \alpha_{jr}} C_r \right)^{\alpha_{ir}} \approx$$

$$\prod_{r=1}^{R} \left(\frac{\alpha'_{ir}}{\sum_{j \neq i} \alpha_{jr}} C_r \right)^{\alpha_{ir}} =$$

$$A_i \prod_{r=1}^{R} \alpha'^{\alpha_{ir}}_{ir},$$

where $A_i = \prod_{r=1}^{R} C_r / \sum_{j \neq i} \alpha_{jr}$ is a constant. Let us consider the following optimization problem:

$$\begin{aligned} &\text{maximize} && \prod_r \alpha_{ir}'^{\alpha_{ir}} \\ &\text{subject to} && \sum_r \alpha_{ir}' = 1. \end{aligned}$$

Now, consider the Lagrangian form:

$$L(\alpha', \lambda) = \prod_r \alpha_{ir}'^{\alpha_{ir}} - \lambda(1 - \sum_r \alpha_{ir}'),$$

where λ is a Lagrange multiplier. Then, based on the KKT conditions:

$$\begin{aligned} \frac{\partial L}{\partial \alpha_{ir}'} &= \alpha_{ir} \frac{\prod_r \alpha_{ir}'^{\alpha_{ir}}}{\alpha_{ir}'} - \lambda = 0 \qquad \forall r \\ \frac{\partial L}{\partial \lambda} &= 1 - \sum_r \alpha_{ir}' = 0 \end{aligned}$$

The solution to these equations is $\alpha_{ir}' = \alpha_{ir}$, for all resources r.

A.2 Proof of Lemma 14

We use the characterization of the FL mechanism allocations from Lemma 13. We consider four cases, corresponding to whether or not supply exceeds demand in the truthful and misreported instances. Let x' denote the objective value in the FL mechanism's call to PSWC in the misreported instance, and x in the truthful instance. Suppose first that $D_r \geq E$ and $D_r' \geq E$. Suppose that $x' \leq x$. Then, for all $j \neq i$,

$$a_{j,r}' = \min(x'e_j, d_{j,r}, t_{j,r}') \leq \min(xe_j, d_{j,r}, t_{j,r}) = a_{j,r},$$

which implies that $a_{i,r}' \geq a_{i,r}$, since $\sum_{k \in [n]} a_{k,r'} = \sum_{k \in [n]} a_{k,r'}'$. On the other hand, if $x' > x$, then

$$a_{i,r}' = \min(x'e_i, d_{i,r}, t_{i,r}') \geq \min(xe_i, d_{i,r}, t_{i,r}) = a_{i,r}.$$

Second, suppose that $D_r \geq E$ and $D'_r < E$. Then

$$a'_{i,r} \geq \min(d_{i,r}, t'_{i,r}) \geq \min(d_{i,r}, t_{i,r}) \geq a_{i,r}.$$

Third, suppose that $D_r < E$ and $D'_r \geq E$. Then

$$a'_{j,r} \leq \min(d_{j,r}, t'_{j,r}) \leq \min(d_{j,r}, t_{j,r}) \leq a_{j,r}$$

for all $j \neq i$, which implies that $a'_{i,r} \geq a_{i,r}$. Finally, suppose that $D_r < E$ and $D'_r < E$. If $x' \leq x$, then for all $j \neq i$, we have that

$$a'_{j,r} = \min(t'_{j,r}, \max(d_{j,r}, x'e_j)) \leq \min(t_{j,r}, \max(d_{j,r}, xe_j)) = a_{j,r},$$

which implies that $a'_{i,r} \geq a_{i,r}$. If $x' > x$, then

$$a'_{i,r} = \min(t'_{i,r}, \max(d_{i,r}, x'e_i)) \geq \min(t_{i,r}, \max(d_{i,r}, xe_i)) = a_{i,r}.$$

Thus, the lemma holds in all cases.

A.3 Proof of Lemma 27

If $D \geq E$, substituting the relevant terms into Lemma 11 gives us the following.

$$a_{i,r} = \max(0, \min(\min(d'_{i,r}, e_i + b_i), xe_i)) = \min(e_i + b_i, d'_{i,r}, xe_i).$$

If $D < E$, then again by substituting into Lemma 11 we have the following.

$$a_{i,r} = \max(\min(e_i + b_i, d'_{i,r}), \min(e_i + b_i, xe_i)) = \min(e_i + b_i, \max(d'_{i,r}, xe_i)).$$

The final equality, $\max(\min(A, B), \min(A, C)) = \min(A, \max(B, C))$, can easily be checked to hold case by case for any relative ordering of A, B, and C.

A.4 Proof of Lemma 28

If $r > T$, then the allocation of agent i is independent of her reported demand, thus $a_{i,r} = a'_{i,r}$. Now suppose that $r \leq T$. Let $\bar{d}_{i,r} = \min(d_{i,r}, e_i + b_{i,r})$ and $\bar{d}'_{i,r} =$

$\min(d'_{i,r}, e_i + b_{i,r})$. Also, let x and x' denote the objective value in the T-period mechanism's call to PSWC when i reports $d_{i,r}$ and $d'_{i,r}$, respectively. Observe first that $D' = \bar{d}'_{i,r} + \sum_{j\neq i} \bar{d}_{j,r} \leq \bar{d}_{i,r} + \sum_{j\neq i} \bar{d}_{j,r} = D$.

Suppose first that $E \leq D' \leq D$. Let $a_{j,r}$ and $a'_{j,r}$ denote the allocations of $j/not = i$ when i reports $d_{i,r}$ and $d'_{i,r}$, respectively. If $x' \geq x$, then for all $j \neq i$, by Lemma 27 we have:

$$a'_{j,r} = \min(e_j + b_{j,r}, d_{j,r}, x'e_j) \geq \min(e_j + b_{j,r}, d_{j,r}, xe_j) = a_{j,r}.$$

This immediately implies that $a'_{i,r} \leq a_{i,r}$, because $\sum_{k\in[n]} a_{k,r} = \sum_{k\in[n]} a'_{k,r} = E$. If $x' < x$, then again by Lemma 27 we have the following:

$$a'_{i,r} = \min(e_i + b_{i,r}, d'_{i,r}, x'e_i) \leq \min(e_i + b_{i,r}, d_{i,r}, xe_i) = a_{i,r}.$$

By the same lemma, for all $j \neq i$, we also have:

$$a'_{j,r} = \min(e_j + b_{j,r}, d_{j,r}, x'e_j) \leq \min(e_j + b_{j,r}, d_{j,r}, xe_j) = a_{j,r}.$$

Therefore, for all $k \in [n]$, $a_{k,r} \geq a'_{k,r}$. However, since $\sum_{k\in[n]} a_{k,r} = \sum_{k\in[n]} a'_{k,r} = E$, it has to be the case that $a_{k,r} = a'_{k,r}$ for all k.

Next, suppose that $D' < E \leq D$. By the definition of the T-period mechanism, for all $j \neq i$, $a'_{j,r} \geq \bar{d}_{j,r}$, and $a_{j,r} \leq \bar{d}_{j,r}$. Therefore, $a'_{j,r} \geq a_{j,r}$ which implies that $a'_{i,r} \leq a_{i,r}$.

Finally, suppose that $D' \leq D < E$. If $x' \geq x$, then by Lemma 27, for all $j \neq i$, we have:

$$a'_{j,r} = \min(e_j + b_{j,r}, \max(d_{j,r}, x'e_j)) \geq \min(e_j + b_j, \max(d_{j,r}, xe_j)) = a_{j,r}.$$

This implies $a'_{i,r} \leq a_{i,r}$. If $x' < x$, then, by Lemma 27 we have:

$$a'_{i,r} = \min(e_i + b_{i,r}, \max(d'_{i,r}, x'e_i)) \leq \min(e_i + b_{i,r}, \max(d_{i,r}, xe_i)) = a_{i,r}$$

By the same lemma, for all $j \neq i$, we also have:

$$a'_{j,r} = \min(e_j + b_{j,r}, \max(d_{j,r}, x'e_j)) \leq \min(e_j + b_{j,r}, \max(d_{j,r}, xe_j)) = a_{j,r}.$$

Therefore, for all $k \in [n]$, $a'_{k,r} \leq a_{k,r}$. However, since $\sum_{k \in [n]} a_{k,r} = \sum_{k \in [n]} a'_{k,r} = E$, it has to be the case that $a_{k,r} = a'_{k,r}$ for all k.

A.5 Proof of Lemma 29

Note that $D' \leq D$, since $d'_{i,r} < d_{i,r}$ and $d'_{j,r} = d_{j,r}$ for all agents $j \neq i$. If $E \leq D$, then by the definition of the T-period mechanism we have:

$$a_{i,r} \leq \bar{d}_{i,r} = \min(e_i + b_i, d_{i,r}) \leq d_{i,r}.$$

Next, assume that $D' \leq D < E$. Then $a'_{i,r} < a_{i,r}$ implies that there is at least one agent j with $a'_{j,r} > a_{j,r}$. In the proof of Lemma 28 we show that if $x' < x$, then $a'_{k,r} = a_{k,r}$ for all k. Therefore, it has to be the case that $x' \geq x$. By Lemma 27, $a_{i,r} = \min(e_i + b_{i,r}, \max(d_{i,r}, xe_i))$ and $a'_{i,r} = \min(e_i + b_i, \max(d'_{i,r}, x'e_i))$. It is easy to see that if $d_{i,r} < xe_i$, then $a'_{i,r} \geq a_{i,r}$, which contradicts the assumption in the lemma statement. Therefore, we have:

$$a_{i,r} = \min(e_i + b_i, \max(d_{i,r}, xe_i)) = \min(e_i + b_i, d_{i,r}) \leq d_{i,r}.$$

A.6 Proof of Corollary 30

Because $a'_{i,r} < a_{i,r} \leq d_{i,r}$, we can substitute the utility values from Equation (6.2),

$$u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r}) = a_{i,r}H - a'_{i,r}H = H(a_{i,r} - a'_{i,r}).$$

A.7 Proof of Lemma 31

Note that $D' \geq D$, since $d'_{i,r} > d_{i,r}$ and $d'_{j,r} = d_{j,r}$ for all agents $j \neq i$. If $D < E$, then $a_{i,r} \geq \bar{d}_{i,r} = \min(e_i + b_{i,r}, d_{i,r})$. We show that $e_i + b_{i,r} \geq d_{i,r}$, and therefore, $a_{i,r} \geq d_{i,r}$. Suppose for contradiction that $e_i + b_{i,r} < d_{i,r} < d'_{i,r}$, which means $\bar{d}_{i,r} = \bar{d}'_{i,r} = e_i + b_{i,r}$. By definition of the T-period mechanism, $\bar{d}_{i,r} \leq a_{i,r} \leq e_i + b_{i,r}$, which implies $a_{i,r} = e_i + b_{i,r}$. Also, by the definition of the mechanism, $a'_{i,r} \leq \bar{d}'_{i,r} = e_i + b_{i,r}$

if $D' \geq E$, and $a'_{i,r} \leq e_i + b'_{i,r} = e_i + b_{i,r}$ if $D' < E$. In both cases, $a'_{i,r} \leq e_i + b_{i,r} = a_{i,r}$, a contradiction to the assumption in the lemma statement.

If $D' \geq D \geq E$, then $a'_{i,r} > a_{i,r}$ implies that there is at least an agent j with $a'_{j,r} < a_{j,r}$. In the proof of Lemma 28 we show that if $x < x'$, then $a'_{k,r} = a_{k,r}$ for all k. Therefore, it has to be the case that $x \geq x'$. By Lemma 27, $a_{i,r} = \min(e_i + b_{i,r}, d_{i,r}, xe_i)$ and $a'_{i,r} = \min(e_i + b_{i,r}, d'_{i,r}, x'e_i)$. It is easy to see that if $a_{i,r}$ is xe_i or $e_i + b_{i,r}$, then $a'_{i,r} \leq a_{i,r}$. Therefore, $a_{i,r} = d_{i,r}$, which means the lemma holds.

A.8 Proof of Corollary 32

Because $d_{i,r} \leq a_{i,r} < a'_{i,r}$, we can substitute the utility values from Equation (6.2),

$$u_{i,r}(a'_{i,r}) - u_{i,r}(a_{i,r}) = d_{i,r}H + (a'_{i,r} - d_{i,r})L - (d_{i,r}H + (a_{i,r} - d_{i,r})L) = L(a'_{i,r} - a_{i,r}).$$

A.9 Proof of Lemma 33

Suppose first that agent i reports $d'_{i,T} < d_{i,T}$. Then, by Lemma 28, $a'_{i,T} \leq a_{i,T}$. If $a'_{i,T} = a_{i,T}$, then the misreport has had no effect on the allocations, since the allocation at rounds $r \leq T$ is unchanged, and the allocations at rounds $r > T$ depend only on the allocations at rounds $r \leq T$, not the reports. So assume that $a'_{i,T} = a_{i,T} - k$ for some $k > 0$. By the definition of the T-Period mechanism, i's allocation increases by $\frac{k}{T}$ for each of rounds $T + 1, \ldots, 2T$. The difference between her utility from truthfully reporting at round T and from misreporting at round T

is given by

$$
\begin{aligned}
U_{i,R}(\mathbf{a_i}) - U_{i,R}(\mathbf{a_i'}) &= \sum_{r=1}^{R} \left(u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r})\right) \\
&= u_{i,T}(a_{i,T}) - u_{i,T}(a'_{i,T}) + \sum_{r=T+1}^{2T} \left(u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r})\right) \\
&= kH + \sum_{r=T+1}^{2T} \left(u_{i,r}(a_{i,r}) - u_{i,r}(a_{i,r} + \frac{k}{T})\right) \geq kH - kH = 0
\end{aligned}
$$

where the second transition follows because $d'_{i,r} = d_{i,r}$ for all rounds $r < T$, the third transition from Corollary 30, and the final transition because each of the extra resources received in the misreported case for rounds $r > T$ can each be worth at most H to i.

Next suppose that agent i reports $d'_{i,T} > d_{i,T}$. Then, by Lemma 28, $a'_{i,T} \geq a_{i,T}$. As before, assume that $a'_{i,T} \neq a_{i,T}$. That is, $a'_{i,T} = a_{i,T} + k$ for some $k > 0$. By the definition of the T-Period mechanism, i's allocation decreases by $\frac{k}{T}$ for each of rounds $T + 1, \ldots, 2T$. The difference between her utility from truthfully reporting at round T and from misreporting at round T is given by

$$
\begin{aligned}
U_{i,R}(\mathbf{a_i}) - U_{i,R}(\mathbf{a_i'}) &= \sum_{r=1}^{R} \left(u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r})\right) \\
&= u_{i,T}(a_{i,T}) - u_{i,T}(a'_{i,T}) + \sum_{r=T+1}^{2T} \left(u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r})\right) \\
&= -kL + \sum_{r=T+1}^{2T} \left(u_{i,r}(a_{i,r}) - u_{i,r}(a_{i,r} - \frac{k}{T})\right) \geq -kL + kL = 0
\end{aligned}
$$

where the second transition follows because $d'_{i,r} = d_{i,r}$ for all rounds $r < T$, the third transition from Corollary 32, and the final transition because each of the extra resources received in the truthful case for rounds $r > T$ are each worth at least L to i.

A.10 Proof of Lemma 35

We treat four cases, corresponding to whether or not supply exceeds demand in the truthful and misreported instances. Let x' denote the objective value in the T-Period mechanism's call to PSWC in the misreported instance, and x in the truthful instance. All cases rely heavily on the characterization of the allocation from Lemma 27.

Suppose first that $D_r \geq E$ and $D'_r \geq E$. Suppose that $x' \leq x$. Then, for all $j \neq i$, $a'_{j,r} = \min(x'e_j, d_{j,r}, e_j + b'_{j,r}) \leq \min(xe_j, d_{j,r}, e_j + b_{j,r}) = a_{j,r}$, which implies that $a'_{i,r} \geq a_{i,r}$, since $\sum_{k\in[n]} a_{k,r'} = \sum_{k\in[n]} a'_{k,r'}$. On the other hand, if $x' > x$, then $a'_{i,r} = \min(x'e_i, d_{i,r}, e_i + b'_{i,r}) \geq \min(xe_i, d_{i,r}, e_i + b_{i,r}) = a_{i,r}$. Second, suppose that $D_r \geq E$ and $D'_r < E$. Then $a'_{i,r} \geq \min(d_{i,r}, e_i + b'_{i,r}) \geq \min(d_{i,r}, e_i + b_{i,r}) \geq a_{i,r})$. Third, suppose that $D_r < E$ and $D'_r \geq E$. Then $a'_{j,r} \leq \min(d_{j,r}, e_j + b'_{j,r}) \leq \min(d_{j,r}, e_j + b_{j,r}) \leq a_{j,r}$ for all $j \neq i$, which implies that $a'_{i,r} \geq a_{i,r}$.

Finally, suppose that $D_r < E$ and $D'_r < E$. If $x' \leq x$, then for all $j \neq i$, we have that $a'_{j,r} = \min(e_j + b'_{j,r}, \max(d_{j,r}, x'e_j)) \leq \min(e_j + b_{j,r}, \max(d_{j,r}, xe_j)) = a_{j,r}$, which implies that $a'_{i,r} \geq a_{i,r}$. If $x' > x$, then $a'_{i,r} = \min(e_i + b'_{i,r}, \max(d_{i,r}, x'e_i)) \geq \min(e_i + b_{i,r}, \max(d_{i,r}, xe_i)) = a_{i,r}$. Thus, the lemma holds in all cases.

A.11 Proof of Proposition 37

Let $r \leq T$. First suppose that $D < E$. Then i's minimum allocation is $\bar{d}_{i,r} = \min(d'_{i,r}, e_i + b_{i,r}) = e_i$. So we know that $a_{i,r} \geq e_i$. Suppose for contradiction that $a_{i,r} > e_i$. Then there must be some agent $j \neq i$ with $a_{j,r} \leq e_j$. But now we could obtain a smaller value of x in the PSWC program by assigning slightly higher allocation to j, and slightly lower allocation to any agent with $a_{k,r}/e_k = x$ (we know that j is not one of these agents since $a_{j,r}/e_j < 1 < a_{i,r}/e_i \leq x$). This contradicts optimality of the PSWC program, therefore $a_{i,r} = e_i$.

Next, suppose that $D \geq E$. Then i's limit allocation is $\bar{d}_{i,r} = \min(d'_{i,r}, e_i + b_{i,r}) = e_i$. So we know that $a_{i,r} \leq e_i$. Suppose for contradiction that $a_{i,r} < e_i$. Then there must exist some agent j with $a_{j,r} > e_j$. But now the objective value x of the call to PSWC could be improved by transferring some small amount of allocation to i from all agents k with $a_{k,r}/e_k = x$ (we know that i is not one of these agents since $a_{i,r}/e_i < 1 < a_{j,r}/e_j \leq x$). This contradicts optimality of the PSWC program, therefore $a_{i,r} = e_i$.

A.12 Over-reporting Demand is not Advantageous

In this section we assume that $d'_{i,r'} > d_{i,r'}$. The setup otherwise mirrors that of Section 6.5.3.

Lemma 40. *For all agents $j \neq i$, we have that $a'_{j,r'} \leq a_{j,r'}$. Further, $a'_{i,r'} \geq a_{i,r'}$.*

Proof. We prove the statement for all $j \neq i$. The statement for i follows immediately because the total number of resources to allocate is fixed.

Observe first that

$$D_{r'} = \sum_{k \in [n]} \min(d_{k,r'}, t_{k,r'}) \leq \sum_{k \in [n]} \min(d'_{k,r'}, t_{k,r'}) = D'_{r'},$$

since i's demand increases in the misreported instances but all other demands and token counts stay the same. Let x' denote the objective value in FL's call to PSWC in the misreported instance, and x in the truthful instance.

Suppose that $E \leq D_{r'} \leq D'_{r'}$. Suppose first that $x' < x$. Then, by Lemma 13,

$$a_{j,r'} = \min(xe_j, d_{j,r'}, t_{j,r'}) \geq \min(x'e_j, d_{j,r'}, t_{j,r'}) = a'_{j,r'}$$

for all $j \neq i$. Next, suppose that $x' \geq x$. Then, again by Lemma 13 and the fact that $d'_{i,r'} > d_{i,r'}$,

$$a'_{i,r'} = \min(x'e_i, d'_{i,r'}, t_{i,r'}) \geq \min(xe_i, d_{i,r'}, t_{i,r'}) = a_{i,r'}.$$

And, for all $j \neq i$,

$$a'_{j,r'} = \min(x'e_j, d_{j,r'}, t_{j,r'}) \geq \min(xe_j, d_{j,r'}, t_{j,r'}) = a_{j,r'}.$$

Because $a'_{k,r'} \geq a_{k,r'}$ for all users k, and $\sum_{k \in [n]} a_{k,r'} = \sum_{k \in [n]} a'_{k,r'}$, it must be the case that $a'_{k,r'} = a_{k,r'}$ for all k, which satisfies the statement of the lemma.

Next, suppose that $D_{r'} < E \leq D'_{r'}$. By the definition of FL, $a_{k,r'} \geq \min(d_{k,r'}, t_{k,r'})$ for all k, and $a'_{k,r'} \leq \min(d'_{k,r'}, t_{k,r'})$ for all k. Since $\min(d'_{j,r'}, t_{j,r'}) = \min(d_{j,r'}, t_{j,r'})$ for all $j \neq i$, we have that $a_{j,r'} \geq a'_{j,r'}$, implying also that $a_{i,r'} \leq a'_{i,r'}$.

Finally, suppose that $D_{r'} \leq D'_{r'} < E$. Suppose first that $x \leq x'$. Then, by Lemma 13 and the assumption that $d_{i,r'} < d'_{i,r'}$, we have

$$a_{i,r'} = \min(t_{i,r'}, \max(xe_i, d_{i,r'})) \leq \min(t_{i,r'}, \max(x'e_i, d'_{i,r'})) = a'_{i,r'}$$

and

$$a_{j,r'} = \min(t_{j,r'}, \max(xe_j, d_{j,r'})) \leq \min(t_{j,r'}, \max(x'e_j, d_{j,r'})) = a'_{j,r'}$$

for all $j \neq i$. Because $a_{k,r'} \leq a'_{k,r'}$ for all users k, and $\sum_{k \in [n]} a'_{k,r'} = \sum_{k \in [n]} a_{k,r'}$, it must be the case that $a_{k,r'} = a'_{k,r'}$ for all k, which satisfies the lemma statement. Next, suppose that $x > x'$. Then, again by Lemma 13, for all $j \neq i$, we have

$$a_{j,r'} = \min(t_{j,r'}, \max(xe_j, d_{j,r'})) \geq \min(t_{j,r'}, \max(x'e_j, d_{j,r'})) = a'_{j,r'}.$$

□

If it is the case that $a'_{i,r'} = a_{i,r'}$, then it must also be that $a'_{j,r'} = a_{j,r'}$ for all $j \neq i$. So allocations at round r' are the same in the misreported instance as the truthful instance. Therefore, for all rounds $r \leq r'$, allocations in both universes are the same. In all rounds $r > r'$, reports in both universes are the same. Together, these imply that allocations for all rounds $r > r'$ are the same in both universes. In particular, i does not profit from her misreport and could weakly improve her utility by reporting $d'_{i,r'} = d_{i,r'}$. So, for the remainder of this section, we assume that $a'_{i,r'} > a_{i,r'}$.

Our next lemma says that the additional resources that i receives in round r' are *low* valued resources for her. The intuition is that if it were the case that i was receiving only high-valued resources under truthful reporting, then she will not receive any extra resources by misreporting (since no agent donates any additional resources for i to receive).

Lemma 41. *If $a'_{i,r'} > a_{i,r'}$, then $a_{i,r'} \geq d_{i,r'}$.*

Proof. Suppose for contradiction that $a_{i,r'} < d_{i,r'}$. We also know that $a_{i,r'} < a'_{i,r'} \leq t'_{i,r'} = t_{i,r'}$, where the equality holds because allocations before round r' are identical in the truthful and misreported instances. It must therefore be the case that $D'_{r'} \geq D_{r'} > E$, where the first inequality holds because $d'_{j,r'} = d_{j,r'}$ for all $j \neq i$ and $d'_{i,r'} > d_{i,r'}$, and the second because $a_{i,r'} < \min(t_{i,r'}, d_{i,r'})$. Let x denote the objective value of FL's call to PSWC in the truthful instance, and x' in the misreported instance. Suppose that $x \leq x'$. Then, by Lemma 13 and the assumption that $d_{i,r'} < d'_{i,r'}$,

$$a_{i,r'} = \min(t_{i,r'}, xe_i, d_{i,r'}) \leq \min(t_{i,r'}, x'e_i, d'_{i,r'}) = a'_{i,r'},$$

and for all $j \neq i$

$$a_{j,r'} = \min(t_{j,r'}, xe_j, d_{j,r'}) \leq \min(t_{j,r'}, x'e_j, d_{j,r'}) = a'_{j,r'}.$$

Because $a_{k,r'} \leq a'_{k,r'}$ for all agents k, and $\sum_{k\in[n]} a_{k,r'} = \sum_{k\in[n]} a'_{k,r'}$, it must be the case that $a'_{k,r'} = a_{k,r'}$ for all k. This contradicts the assumption that $a_{i,r'} < a'_{i,r'}$.

Now suppose that $x > x'$. Note that $xe_i < d_{i,r'} < d'_{i,r'}$, where the first inequality holds because $a_{i,r'} < \min(t_{i,r'}, d_{i,r'})$. Then, again by Lemma 13 and the previous observation, we have

$$a'_{i,r'} = \min(t_{i,r'}, x'e_i, d'_{i,r'}) \leq \min(t_{i,r'}, xe_i, d_{i,r'}) = a_{i,r'},$$

which contradicts that $a_{i,r'} < a'_{i,r'}$.

Since we arrive at a contradiction in all cases, the lemma statement must be true. □

As a corollary, we can write the difference in utility between the truthful and misreported instances that i derives from round r'.

Corollary 42. $u_{i,r'}(a'_{i,r'}) - u_{i,r'}(a_{i,r'}) = L(a'_{i,r'} - a_{i,r'})$.

Proof. Because $d_{i,r'} \leq a_{i,r'} < a'_{i,r'}$, we can substitute the utility values from Equation (6.2):

$$u_{i,r'}(a'_{i,r'}) - u_{i,r'}(a_{i,r'}) = d_{i,r'}H + (a'_{i,r'} - d_{i,r'})L - d_{i,r'}H - (a'_{i,r} - d_{i,r'})L = L(a'_{i,r'} - a_{i,r'}).$$

□

For a fixed agent k, denote by r'_k the round at which agent k runs out of tokens in the misreported universe. That is, r'_k is the first (and only) round with $a'_{r_k} = t'_{k,r_k} > 0$. Note that $r'_i \geq r'$, since $a'_{i,r'} > 0$. Given this, our next lemma states that, under certain conditions, the effect of i's misreport, $d'_{i,r} > d_{i,r}$, is to decrease the objective value of FL's call to PSWC.

Lemma 43. *Let $r < r'_i$ (i.e.$a'_{i,r} < t'_{i,r}$). Suppose $t_{j,r} \leq t'_{j,r}$ for all agents $j \neq i$. Suppose that either $\min(D_r, D'_r) \geq E$ or $\max(D_r, D'_r) < E$. Then $x' \leq x$, where x' denotes the objective value of FL's call to PSWC in the misreported instance and x in the truthful instance.*

Proof. First, suppose that $\min(D_r, D'_r) \geq E$. Suppose for contradiction that $x < x'$. By Lemma 13,

$$a_{j,r} = \min(xe_j, d_{j,r}, t_{j,r}) \leq \min(x'e_j, d_{j,r}, t'_{j,r}) = a'_{j,r}$$

for all $j \neq i$, where the inequality follows from the assumption that $x < x'$ and that $t_{j,r} \leq t'_{j,r}$. Further,

$$\begin{aligned} a_{i,r} &= \min(xe_i, d_{i,r}, t_{i,r}) \leq \min(xe_i, d_{i,r}) \leq \min(x'e_i, d_{i,r}) \\ &= \min(x'e_i, d_{i,r}, t'_{i,r}) = a'_{i,r}, \end{aligned}$$

where the second inequality follows from the assumption that $x < x'$ and the second to last equality from the assumption $a'_{i,r} < t'_{i,r}$.

Therefore, $a_{k,r} \leq a'_{k,r}$ for all agents k. Since $\sum a'_{k,r} = \sum a_{k,r}$, it must be the case that $a'_{k,r} = a_{k,r}$ for all agents k. Therefore, by the definition of FL, $a'_{k,r}/e_k \leq x < x'$ for all agents k with $a'_{k,r} > m_k = 0$. Therefore x' is not the optimal objective value of the PSWC program in the misreported instance, a contradiction. Thus, $x \geq x'$.

Next, suppose that $\max(D_r, D'_r) < E$. Suppose for contradiction that $x < x'$. By Lemma 13,

$$a_{j,r} = \min(t_{j,r}, \max(xe_j, d_{j,r})) \leq \min(t'_{j,r}, \max(x'e_j, d_{j,r})) = a'_{j,r}$$

for all $j \neq i$, where the inequality follows from the assumption that $x < x'$ and that $t_{j,r} \leq t'_{j,r}$. Further,

$$\begin{aligned} a_{i,r} &= \min(t_{i,r}, \max(xe_i, d_{i,r})) \leq \max(xe_i, d_{i,r}) \leq \max(x'e_i, d_{i,r}) \\ &= \min(t'_{i,r}, \max(x'e_i, d_{i,r})) = a'_{i,r}, \end{aligned}$$

where the second inequality follows from the assumption that $x < x'$ and the second to last equality from the assumption $a'_{i,r} < t'_{i,r}$.

Therefore, $a_{k,r} \leq a'_{k,r}$ for all agents k. Since $\sum a'_{k,r} = \sum a_{k,r}$, it must be the case that $a'_{k,r} = a_{k,r}$ for all agents k. Consider all agents with $\min(d_{k,r}, t'_{k,r}) < a'_{k,r}$ (that is, those agents for which the first constraint in the PSWC program binds in the misreported instance). For all such agents, we have

$$\begin{aligned} \min(d_{k,r}, t'_{k,r}) < a'_{k,r} &\implies d_{k,r} < a'_{k,r} \leq t'_{k,r} \implies d_{k,r} < a_{k,r} \leq t_{k,r} \\ &\implies \min(d_{k,r}, t_{k,r}) < a_{k,r}, \end{aligned}$$

so the constraints bind in the truthful instance as well. Therefore, $a_{k,r}/e_k \leq x < x'$ for all agents k for which the first constraint binds in the misreported instance. Therefore x' is not the optimal objective value of the PSWC program in the misreported instance, a contradiction. Thus, $x \geq x'$. □

Using Lemma 43, we show our main lemma. It allows us to make an inductive argument that, after gaining some extra resources in round r', i's allocation is (weakly) smaller for all other rounds in the mireported instance than the truthful instance.

Lemma 44. *Let $r' < r < r'_i$ (that is, $a'_{i,r} < t'_{i,r}$). Suppose that $t_{j,r} \leq t'_{j,r}$ for all agents $j \neq i$. Then for all $j \neq i$, either: (1) $a_{j,r} = t_{j,r}$, or (2) $a_{j,r} \geq a'_{j,r}$.*

Proof. Note that $t_{j,r} \leq t'_{j,r}$ for all $j \neq i$ implies that $t_{i,r} \geq t'_{i,r}$, which we use in the proof. Also, because $r' < r$, we know that $d'_{i,r} = d_{i,r}$, as r' is the last round for which $d'_{i,r} \neq d_{i,r}$. We assume that condition 1) from the lemma statement is false ($i.e. a_{j,r} < t_{j,r}$) and show that condition 2) must hold. Suppose first that $D'_r < E$. Then, because $a'_{i,r} < t'_{i,r}$, we know that $d'_{i,r} \leq t'_{i,r} \leq t_{i,r}$. This implies that $\min(d_{i,r}, t'_{i,r}) = \min(d_{i,r}, t_{i,r}) = d_{i,r}$. Let $j \neq i$. Since $t_{j,r} \leq t'_{j,r}$, we have $\min(d_{j,r}, t_{j,r}) \leq \min(d_{j,r}, t'_{j,r})$. Therefore, it is the case that $D_r \leq D'_r < E$. By Lemma 13 and the assumption that $a_{j,r} < t_{j,r}$, it must be the case that $a_{j,r} = \max(d_{j,r}, xe_j)$. Further, by Lemma 43, we know that $x \geq x'$. Therefore, we have

$$a'_{j,r} = \max(d_{j,r}, x'e_j) \leq \max(d_{j,r}, xe_j) = a_{j,r}.$$

That is, condition (2) from the lemma statement holds.

Now suppose that $D'_r \geq E$. Then, from the definition of the mechanism, we have that $a'_{j,r} \leq \min(d_{j,r}, t'_{j,r}) \leq d_{j,r}$. If it is the case that $D_r < E$ then we have that $a_{j,r} \geq \min(d_{j,r}, t_{j,r}) = d_{j,r}$, where the equality holds because otherwise we would have $a_{j,r} \geq \min(d_{j,r}, t_{j,r}) = t_{j,r}$, violating the assumption that $a_{j,r} < t_{j,r}$. Using these inequalities, we have $a_{j,r} \geq d_{j,r} \geq a'_{j,r}$, so condition (2) from the statement of the lemma holds. Finally, it may be the case that $D'_r \geq M$ and $D_r \geq M$. By Lemma 13 and the assumption that $a_{j,r} < t_{j,r}$, we have

$$a_{j,r} = \min(d_{j,r}, xe_k) \geq \min(d_{j,r}, x'e_k) = a'_{j,r},$$

where the inequality follows from Lemma 43. Thus, condition (2) of the lemma statement holds. □

We now show an analogous result to Lemma 14.

Lemma 45. *Suppose that $t_{j,r} \leq t'_{j,r}$ for all $j \neq i$, and $d_{k,r} = d'_{k,r}$ for all $k \in [n]$. Then $a_{i,r} \geq a'_{i,r}$.*

Proof. Note that the condition that $t_{j,r} \leq t'_{j,r}$ for all $j \neq i$ implies that $t_{i,r} \geq t'_{i,r}$. We use these assumptions, along with the characterization of the FL mechanism allocations from Lemma 13, to prove the lemma.

We treat four cases, corresponding to whether or not supply exceeds demand in the truthful and misreported instances. Let x' denote the objective value in the FL mechanism's call to PSWC in the misreported instance, and x in the truthful instance. Suppose first that $D'_r \geq E$ and $D_r \geq E$. Suppose that $x \leq x'$. Then, for all $j \neq i$, $a_{j,r} = \min(xe_j, d_{j,r}, t_{j,r}) \leq \min(x'e_j d_{j,r}, t'_{j,r}) = a'_{j,r}$, which implies that $a_{i,r} \geq a'_{i,r}$, since $\sum_{k\in[n]} a'_{k,r'} = \sum_{k\in[n]} a_{k,r'}$. On the other hand, if $x > x'$, then $a_{i,r} = \min(xe_i, d_{i,r}, t_{i,r}) \geq \min(x'e_i, d_{i,r}, t'_{i,r}) = a'_{i,r}$. Second, suppose that $D'_r \geq E$ and $D_r < E$. Then $a_{i,r} \geq \min(d_{i,r}, t_{i,r}) \geq \min(d_{i,r}, t'_{i,r}) \geq a'_{i,r})$. Third, suppose that $D'_r < E$ and $D_r \geq E$. Then $a_{j,r} \leq \min(d_{j,r}, t_{j,r}) \leq \min(d_{j,r}, t'_{j,r}) \leq a'_{j,r}$ for all $j \neq i$, which implies that $a_{i,r} \geq a'_{i,r}$.

Finally, suppose that $D'_r < E$ and $D_r < E$. If $x \leq x'$, then for all $j \neq i$, we have that $a_{j,r} = \min(t_{j,r}, \max(d_{j,r}, xe_j)) \leq \min(t'_{j,r}, \max(d_{j,r}, x'e_j)) = a'_{j,r}$, which implies that $a_{i,r} \geq a'_{i,r}$. If $x > x'$, then $a_{i,r} = \min(t_{i,r}, \max(d_{i,r}, xe_i)) \geq \min(t'_{i,r}, \max(d_{i,r}, x'e_i)) = a'_{i,r}$. Thus, the lemma holds in all cases. □

Finally, we show that the mechanism is strategy-proof.

Theorem 46. *Agent i never benefits from reporting $d_{i,r'} > d_{i,r'}$.*

Proof. We first observe that for every $r \leq r'_i$, $t_{j,r} \leq t'_{j,r}$ for every $j \neq i$. This is true for every $r \leq r'$ because $a'_{j,r} = a_{j,r}$ for $r < r'$, by Lemma 15. For $r = r' + 1$, it follows from Lemma 40, which says that $a_{j,r'} \geq a'_{j,r'}$. For all subsequent rounds, up to and including $r = r'_i$, it follows inductively from Lemma 44: $t_{j,r} \leq t'_{j,r}$ implies that either $a_{j,r} = t_{j,r}$ (in which case $t_{j,r+1} = 0 \leq t'_{j,r+1}$), or $a_{j,r} \geq a'_{j,r}$ (in which case $t_{j,r+1} = t_{j,r} - a_{j,r} \leq t'_{j,r} - a'_{j,r} = t'_{j,r+1}$).

Consider an arbitrary round $r \neq r'$, with $r \leq r'_i$. By the above argument, we know that $t_{j,r} \leq t'_{j,r}$ for all $j \neq i$. Further, because reports in the truthful and misreported instances are identical on all rounds $r \neq r'$, we have that $d_{k,r} = d'_{k,r}$ for all $k \in [n]$. Therefore, by Lemma 45, $a_{i,r} \geq a'_{i,r}$. For rounds $r > r'_i$, it is also true that $a_{i,r} \geq a'_{i,r}$, since $a'_{i,r} = 0$ for these rounds by the definition of r'_i.

Finally,

$$\begin{aligned}
U_{i,R}(\mathbf{a_i}) - U_{i,R}(\mathbf{a'_i}) &= \sum_{r=1}^{R}(u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r})) \\
&= \sum_{r \neq r'}(u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r})) + (u_{i,r'}(a_{i,r'}) - u_{i,r'}(a'_{i,r'})) \\
&= \sum_{r \neq r'}(u_{i,r}(a_{i,r}) - u_{i,r}(a'_{i,r})) - L(a'_{i,r'} - a_{i,r'}) \\
&\geq L(a'_{i,r'} - a_{i,r'}) - L(a'_{i,r'} - a_{i,r'}) = 0
\end{aligned}$$

Where the third transition follows from Corollary 42, and the final transition because $\sum_{r \neq r'}(a'_{i,r} - a_{i,r}) = a_{i,r'} - a'_{i,r'}$, and every term in the sum is positive. □

Bibliography

[1] S. M. Zahedi and B. C. Lee, "REF: Resource elasticity fairness with sharing incentives for multiprocessors," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014, pp. 145–160.

[2] S. Fan, S. M. Zahedi, and B. C. Lee, "The computational sprinting game," in *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016, pp. 561–575.

[3] S. M. Zahedi, S. Fan, and B. C. Lee, "Managing heterogeneous datacenters with tokens," *ACM Transactions on Architecture and Code Optimization (TACO)*, 2018.

[4] S. M. Zahedi, Q. Llull, and B. C. Lee, "Amdahl's Law in the datacenter era: A market for fair processor allocation," in *Proceedings of the 24rd IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2018.

[5] R. Freeman, S. M. Zahedi, V. Conitzer, and B. C. Lee, "Dynamic proportional sharing: A game-theoretic approach," in *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2018.

[6] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, 2007.

[7] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bit torrent," in *Proceedings of the 4th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2007, pp. 1–1.

[8] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in

Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI), 2011, pp. 323–336.

[9] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, “Large-scale cluster management at google with borg,” in *Proceedings of the 10th European Conference on Computer Systems (EuroSys)*, 2015, p. 18.

[10] S. M. Zahedi and B. C. Lee, “Sharing incentives and fair division for multiprocessors,” *IEEE Micro*, vol. 35, no. 3, pp. 92–100, 2015.

[11] S. M. Zahedi, S. Fan, M. Faw, E. Cole, and B. C. Lee, “Computational sprinting: Architecture, dynamics, and strategies,” *ACM Transactions on Computer Systems (TOCS)*, vol. 34, no. 4, pp. 12:1–12:26, January 2017.

[12] L. A. Barroso and U. Hölzle, “The datacenter as a computer: An introduction to the design of warehouse-scale machines,” *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–108, 2009.

[13] R. Gabor, S. Weiss, and A. Mendelson, “Fairness enforcement in switch on event multithreading,” *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 4, no. 3, p. 15, 2007.

[14] O. Mutlu and T. Moscibroda, “Stall-time fair memory access scheduling for chip multiprocessors,” in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2007, pp. 146–160.

[15] H. Varian, “Equity, envy, and efficiency,” *Journal of economic theory*, vol. 9, no. 1, pp. 63–91, 1974.

[16] K. Hashimoto, “Strategy-proofness versus efficiency on the Cobb-Douglas domain of exchange economies,” *Social choice and welfare*, vol. 31, no. 3, pp. 457–473, 2008.

[17] S. Eyerman and L. Eeckhout, “System-level performance metrics for multiprogram workloads,” *IEEE Micro*, vol. 28, no. 3, pp. 42–53, 2008.

[18] R. Gabor, S. Weiss, and A. Mendelson, “Fairness and throughput in switch on event multithreading,” in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2006, pp. 149–160.

[19] R. Bitirgen, E. Ipek, and J. F. Martinez, “Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach,”

in *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2008, pp. 318–329.

[20] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2012, pp. 1206–1214.

[21] D. Dolev, D. G. Feitelson, J. Y. Halpern, R. Kupferman, and N. Linial, "No justified complaints: On fair sharing of multiple resources," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS)*, 2012, pp. 68–75.

[22] M. D. Hill and M. R. Marty, "Amdahl's Law in the multicore era," *Computer*, vol. 41, no. 7, 2008.

[23] A. Toriello and J. P. Vielma, "Fitting piecewise linear continuous functions," *European Journal of Operational Research*, vol. 219, no. 1, pp. 86–95, 2012.

[24] F. Y. Edgeworth, *Mathematical psychics: An essay on the application of mathematics to the moral sciences*. C. Kegan Paul & Company, 1881, no. 10.

[25] A. Muthoo, *Bargaining theory with applications*. Cambridge University Press, 1999.

[26] J. F. Nash Jr, "The bargaining problem," *Econometrica: Journal of the Econometric Society*, pp. 155–162, 1950.

[27] H. Moulin, *Fair division and collective welfare*. MIT press, 2004.

[28] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011, pp. 248–259.

[29] A. Patel, F. Afram, S. Chen, and K. Ghose, "Marss: A full system simulator for multicore x86 CPUs," in *Proceedings of the 48th Design Automation Conference (DAC)*, 2011, pp. 1050–1055.

[30] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[31] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*, vol. 19, no. 4, 1989, pp. 1–12.

[32] C. A. Waldspurger and W. E. Weihl, "Lottery Scheduling: Flexible proportional-share resource management," in *Proceedings of the 1st Symposium on Operating Systems Design and Implementation (OSDI)*, 1994, pp. 1–11.

[33] O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems," in *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA)*, 2008, pp. 63–74.

[34] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," *Optimization and Engineering*, vol. 8, no. 1, pp. 67–127, 2007.

[35] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 1.21," 2010.

[36] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[37] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating MapReduce for multi-core and multiprocessor systems," in *Proceedings of the 13th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2007, pp. 13–24.

[38] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2006, pp. 185–194.

[39] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008, pp. 72–81.

[40] D. C. Parkes, A. D. Procaccia, and N. Shah, "Beyond dominant resource fairness: Extensions, limitations, and indivisibilities," in *Proceedings of the 13th ACM Conference on Electronic Commerce (EC)*, 2012, pp. 808–825.

[41] A. Gutman and N. Nisan, "Fair allocation without trade," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012, pp. 719–728.

[42] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith, "Fair queuing memory systems," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2006, pp. 208–222.

[43] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Fairness via source throttling: A configurable and high-performance fairness substrate for multi-core memory systems," in *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2010, pp. 335–346.

[44] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das, "Application-aware prioritization mechanisms for on-chip networks," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 280–291.

[45] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *Proceedings of the IEEE 16th International Symposium on High Performance Computer Architecture (HPCA)*, 2010, pp. 1–12.

[46] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, 2001, pp. 103–116.

[47] B. Lubin, J. O. Kephart, R. Das, and D. C. Parkes, "Expressive power-based resource allocation for data centers." in *Proceedings of the 21st International Joint Conferences on Artificial Intelligence (IJCAI)*, 2009, pp. 1451–1456.

[48] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA)*, 2007, pp. 13–23.

[49] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. K. Martin, "Computational sprinting," in *Proceedings of the 18th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2012, pp. 1–12.

[50] A. Raghavan, L. Emurian, L. Shao, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. Martin, "Computational sprinting on a hardware/software testbed," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013, pp. 155–166.

[51] W. Zheng and X. Wang, "Data center sprinting: Enabling computational sprinting at the data center level," in *Proceedings of the 35th International Conference on Distributed Computing Systems (ICDCS)*, 2015, pp. 175–184.

[52] M. Skach, M. Arora, C.-H. Hsu, Q. Li, D. Tullsen, L. Tang, and J. Mars, "Thermal time shifting: Leveraging phase change materials to reduce cooling costs in warehouse-scale computers," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 439–449.

[53] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*, vol. 10, 2010, p. 10.

[54] L. Shao, A. Raghavan, L. Emurian, M. C. Papaefthymiou, T. F. Wenisch, M. M. Martin, and K. P. Pipe, "On-chip phase change heat sinks designed for computational sprinting," in *Proceedings of the 30th Annual Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*, 2014, pp. 29–34.

[55] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron, "CMP design space exploration subject to physical constraints," in *Proceedings of the 12th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2006, pp. 17–28.

[56] A. Raghavan, L. Emurian, L. Shao, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. Martin, "Utilizing dark silicon to save energy with computational sprinting," *IEEE Micro*, vol. 33, no. 5, pp. 20–28, 2013.

[57] F. Volle, S. V. Garimella, M. Juds *et al.*, "Thermal management of a soft starter: Transient thermal impedance model and performance enhancements using phase change materials," *Power Electronics, IEEE Transactions on*, vol. 25, no. 6, pp. 1395–1405, 2010.

[58] A. Raghavan, "Computational sprinting: Exceeding sustainable power in thermally constrained systems," Ph.D. dissertation, University of Pennsylvania, 2013.

[59] C.-H. Hsu, Y. Zhang, M. Laurenzano, D. Meisner, T. Wenisch, J. Mars, L. Tang, R. G. Dreslinski *et al.*, "Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting," in *Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 271–282.

[60] X. Fu, X. Wang, and C. Lefurgy, "How much power oversubscription is safe and allowed in data centers," in *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC)*, 2011, pp. 21–30.

[61] X. Wang, M. Chen, C. Lefurgy, and T. Keller, "SHIP: Scalable hierarchical power control for large-scale data centers," in *Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2009, pp. 91–100.

[62] Allen-Bradley, "Bulletin 1489 UL489 circuit breakers," URL http://literature.rockwellautomation.com/idc/groups/literature/documents/td/1489-td001_-en-p.pdf, 2016, online; accessed: 12-29-2016.

[63] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar, "Benefits and limitations of tapping into stored energy for datacenters," in *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 341–351.

[64] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar, "Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters," in *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012, pp. 75–86.

[65] Ametek, "Selection and sizing of batteries for UPS backup," URL http://www.solidstatecontrolsinc.com/knowledgecenter/~/media/85b8e51754c446bda1f38449f444471c.ashx, 2016, online; accessed: 12-29-2016.

[66] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2011, pp. 295–308.

[67] J.-M. Lasry and P.-L. Lions, "Mean field games," *Japanese journal of mathematics*, vol. 2, no. 1, pp. 229–260, 2007.

[68] S. Adlakha and R. Johari, "Mean field equilibrium in dynamic games with strategic complementarities," *Operations Research*, vol. 61, no. 4, pp. 971–989, 2013.

[69] S. Adlakha, R. Johari, and G. Y. Weintraub, "Equilibria of dynamic games with many players: Existence, approximation, and market structure," *Journal of Economic Theory*, 2013.

[70] S. Adlakha, R. Johari, G. Y. Weintraub, and A. Goldsmith, "On oblivious equilibrium in large population stochastic games," in *Proceedings of the 49th IEEE Conference on Decision and Control (CDC).* IEEE, 2010, pp. 3117–3124.

[71] K. Iyer, R. Johari, and M. Sundararajan, "Mean field equilibria of dynamic auctions with learning," *ACM SIGecom Exchanges*, vol. 10, no. 3, pp. 10–14, 2011.

[72] R. Gummadi, R. Johari, and J. Y. Yu, "Mean field equilibria of multiarmed bandit games," in *Proceedings of the 13th ACM Conference on Electronic Commerce (EC)*, 2012, pp. 655–655.

[73] J. Stamper, A. Niculescu-Mizil, S. Ritter, G. Gordon, and K. Koedinger, "Algebra I 2006-2007. Challenge data set from KDD Cup 2010 educational data mining challenge," URL http://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp, 2010.

[74] M. Lichman, "UCI machine learning repository," URL http://archive.ics.uci.edu/ml, 2013.

[75] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 5, no. 4, p. 19, 2015.

[76] R. Meusel, S. Vigna, O. Lehmberg, and C. Bizer, "Web data commons - hyperlink graphs," URL http://webdatacommons.org/hyperlinkgraph/index.html, 2012, online; accessed: 12-29-2016.

[77] Z. Mwaikambo, A. Raj, R. Russell, J. Schopp, and S. Vaddagiri, "Linux kernel hotplug CPU support," in *Linux Symposium*, vol. 2, 2004.

[78] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *Proceedings of the 7th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2001, pp. 171–182.

[79] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management," in *Proceedings of the 8th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2002, pp. 17–28.

[80] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008, pp. 337–350.

[81] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proceedings of the ACM SIGCOMM Conference (SIGCOMM).* ACM, 2011, pp. 242–253.

[82] A. A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar, "The need for speed and stability in data center power capping," *Sustainable Computing: Informatics and Systems*, vol. 3, no. 3, pp. 183–193, 2013.

[83] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[84] M. Lin, A. Wierman, L. L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 5, pp. 1378–1391, 2013.

[85] J. L. Berral, Í. Goiri, R. Nou, F. Julià, J. Guitart, R. Gavaldà, and J. Torres, "Towards energy-aware scheduling in data centers using machine learning," in *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking (e-Energy)*, 2010, pp. 215–224.

[86] Í. Goiri, T. D. Nguyen, R. Bianchini, and Í. G. Presa, "CoolAir: Temperature- and variation-aware management for free-cooled datacenters," in *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015, pp. 253–265.

[87] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA)*, 2014, pp. 301–312.

[88] T. Somu Muthukaruppan, A. Pathania, and T. Mitra, "Price theory based power management for heterogeneous multi-cores," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* ACM, 2014, pp. 161–176.

[89] Z. Liu, A. Wierman, Y. Chen, B. Razon, and N. Chen, "Data center demand response: Avoiding the coincident peak via workload shifting and local generation," *Performance Evaluation*, vol. 70, no. 10, pp. 770–791, 2013.

[90] M. Le Treust and S. Lasaulce, "A repeated game formulation of energy-efficient decentralized power control," *Wireless Communications, IEEE Transactions on*, vol. 9, no. 9, pp. 2860–2869, 2010.

[91] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2003, pp. 81–92.

[92] B. C. Lee and D. M. Brooks, "Illustrative design space studies with microarchitectural regression models," in *Proceedings of the 13th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2007, pp. 340–351.

[93] J. Mars and L. Tang, "Whare-Map: Heterogeneity in homogeneous warehouse-scale computers," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013, pp. 619–630.

[94] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013, pp. 77–88.

[95] R. Nathuji, C. Isci, and E. Gorbatov, "Exploiting platform heterogeneity for power efficient data centers," in *Proceedings of the 4th International Conference on Autonomic Computing (ICAC)*, 2007.

[96] V. Janapa Reddi, B. C. Lee, T. Chilimbi, and K. Vaid, "Web search using mobile cores: Quantifying and mitigating the price of efficiency," in *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA)*, 2010, pp. 314–325.

[97] E. Ipek, M. Kirman, N. Kirman, and J. F. Martinez, "Core fusion: Accommodating software diversity in chip multiprocessors," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA)*, 2007, pp. 186–197.

[98] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. Dreslinski, T. F. Wenisch, and S. Mahlke, "Composite cores: Pushing heterogeneity into a core," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012, pp. 317–328.

[99] K. Khubaib, M. A. Suleman, M. Hashemi, C. Wilkerson, and Y. N. Patt, "Morphcore: An energy-efficient microarchitecture for high performance ILP and high throughput TLP," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012, pp. 305–316.

[100] X. Fu, X. Wang, and C. Lefurgy, "How much power oversubscription is safe and allowed in data centers," in *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC)*, 2011, pp. 21–30.

[101] M. L. Littman, T. L. Dean, and L. P. Kaelbling, "On the complexity of solving Markov decision problems," in *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, 1995, pp. 394–402.

[102] "Intel 64 and IA-32 architectures softawre developer's manual." [Online]. Available: https://www-ssl.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf

[103] C. Lucchese, S. Orlando, R. Perego, and F. Silvestri, "WebDocs: A real-life huge transactional dataset," in *Workshop on Frequent Itemset Mining Implementations*, 2004.

[104] D. R. Hower, H. W. Cain, and C. A. Waldspurger, "PABST: Proportional allocation of bandwidth at the source and target," in *Proceedings of the 23rd IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017.

[105] D. Sanchez and C. Kozyrakis, "Vantage: Scalable and efficient fine-grain cache partitioning," in *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 57–68.

[106] R. Manikantan, K. Rajan, and R. Govindarajan, "Probabilistic shared cache management (PriSM)," in *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 428–439.

[107] X. Wang, S. Chen, J. Setter, and J. F. Mart'inez, "SWAP: Effective fine-grain management of shared last-level caches with minimum hardware support," in *Proceedings of the 23rd IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017.

[108] X. Wang and J. F. Martínez, "XChange: A market-based approach to scalable dynamic multi-resource allocation in multicore architectures," in *Proceedings*

of the 21st IEEE International Symposium on High Performance Computer Architecture (HPCA), 2015, pp. 113–125.

[109] K. Van Craeynest, S. Akram, W. Heirman, A. Jaleel, and L. Eeckhout, "Fairness-aware scheduling on single-ISA heterogeneous multi-cores," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2013, pp. 177–187.

[110] A. Gorokh, S. Banerjee, and K. Iyer, "Near-efficient allocation using artificial currency in repeated settings (Extended Abstract)," in *Proceedings of the 12th International Conference on Web and Internet Economics (WINE)*, 2016.

[111] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," in *Proceedings of the ACM SIGCOMM Conference (SIGCOMM)*, 2012, pp. 1–12.

[112] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, 1993.

[113] E. J. Friedman, J. Y. Halpern, and I. Kash, "Efficiency and Nash equilibria in a scrip system for P2P networks," in *Proceedings of the 7th ACM Conference on Electronic Commerce (EC)*, 2006, pp. 140–149.

[114] I. A. Kash, E. J. Friedman, and J. Y. Halpern, "Optimizing scrip systems: Efficiency, crashes, hoarders, and altruists," in *Proceedings of the 8th ACM conference on Electronic Commerce (EC)*, 2007, pp. 305–315.

[115] ——, "Manipulating scrip systems: Sybils and collusion," in *Proceedings of the International Conference on Auctions, Market Mechanisms and Their Applications*, 2009, pp. 13–24.

[116] L. Buttyan and J.-P. Hubaux, "Nuglets: A virtual currency to stimulate cooperation in self-organized mobile ad hoc networks," Tech. Rep., 2001.

[117] J. Xu and M. Van Der Schaar, "Token system design for autonomic wireless relay networks," *IEEE Transactions on Communications*, vol. 61, no. 7, pp. 2924–2935, 2013.

[118] C. Shen, J. Xu, and M. van der Schaar, "Silence is gold: Strategic interference mitigation using tokens in heterogeneous small cell networks," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 6, pp. 1097–1111, 2015.

[119] M. Andrews, L. Qian, and A. Stolyar, "Optimal utility based multi-user throughput allocation subject to throughput constraints," in *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 4, 2005, pp. 2415–2424.

[120] J. Chen and L. K. John, "Efficient program scheduling for heterogeneous multicore processors," in *Proceedings of the 46th Annual Design Automation Conference (DAC)*, 2009, pp. 927–930.

[121] D. Koufaty, D. Reddy, and S. Hahn, "Bias scheduling in heterogeneous multicore architectures," in *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*, 2010, pp. 125–138.

[122] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," in *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012.

[123] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, "Ensemble-level power management for dense blade servers," in *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA)*, 2006, pp. 66–77.

[124] C. Lefurgy, X. Wang, and M. Ware, "Server-level power control," in *Proceedings of the 4th International Conference on Autonomic Computing (ICAC)*, 2007.

[125] M. E. Femal and V. W. Freeh, "Boosting data center performance through nonuniform power allocation," in *Proceedings of the 2nd International Conference on Automatic Computing (ICAC)*, 2005, pp. 250–261.

[126] H. Lim, A. Kansal, and J. Liu, "Power budgeting for virtualized data centers," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, 2011.

[127] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "power" struggles: Coordinated multi-level power management for the data center," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008, pp. 48–59.

[128] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[129] R. Nathuji and K. Schwan, "VPM tokens: Virtual machine-aware power budgeting in datacenters," in *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, 2008, pp. 119–128.

[130] N. Morris, S. M. Renganathan, C. Stewart, R. Birke, and L. Chen, "Sprint ability: How well does your software exploit bursts in processing capacity?" in *Proceedings of the 13th IEEE International Conference on Autonomic Computing (ICAC)*, 2016, pp. 173–178.

[131] J. Kay and P. Lauder, "A fair share scheduler," *Communications of the ACM*, vol. 31, no. 1, pp. 44–55, 1988.

[132] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proceedings of the 10th European Conference on Computer Systems (EuroSys)*, 2015, pp. 18:1–18:17.

[133] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2014, pp. 285–300.

[134] "Amazon EC2," https://aws.amazon.com/ec2/.

[135] "Microsoft azure," https://azure.microsoft.com.

[136] G. J. Henry, "The UNIX system: The fair share scheduler," *AT&T Bell Laboratories Technical Journal*, vol. 63, no. 8, pp. 1845–1857, 1984.

[137] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring), 1967, pp. 483–485.

[138] X. Wang and J. F. Martínez, "ReBudget: Trading off efficiency vs. fairness in market-based multicore resource allocation via runtime budget reassignment," in *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016, pp. 19–32.

[139] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir, "The rise of RaaS: The resource-as-a-service cloud," *Communications of the ACM*, vol. 57, no. 7, pp. 76–84, jul 2014.

[140] ——, "Deconstructing Amazon EC2 spot instance pricing," *ACM Transactions on Economics and Computation*, vol. 1, no. 3, pp. 16:1–16:20, sep 2013.

[141] M. Ben-Yehuda, O. Agmon Ben-Yehuda, and D. Tsafrir, "The nom profit-maximizing operating system," in *Proceedings of the 12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, ser. VEE '16, 2016, pp. 145–160.

[142] S. H. Clearwater and S. D. Kleban, "ASCI Queuing systems: Overview and comparisons," in *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS)*, 2002, p. 49.

[143] J. A. Ang, R. A. Ballance, L. A. Fisk, J. R. Johnston, and K. T. Pedretti, "Red storm capability computing queuing policy," *Cray Users Group*, 2005.

[144] D. University, "The Duke Computer Cluster," http://rc.duke.edu/the-duke-compute-cluster.edu, 2017.

[145] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "HUG: Multi-resource fairness for correlated and elastic demands," in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation (NSDI)*, 2016, pp. 407–424.

[146] D. Wang, D. Irani, and C. Pu, "Evolutionary study of web spam: Webb Spam Corpus 2011 versus Webb Spam Corpus 2006," in *Proceedings of the 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, 2012, pp. 40–49.

[147] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP)*, 2013, pp. 69–84.

[148] A. Karp and H. Flatt, "Measuring parallel processor performance," *Communications of the ACM*, 1990.

[149] "Docker," http:/docs.docker.com.

[150] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation (NSDI)*, 2016, pp. 363–378.

[151] "Tycoon: Market-based resource allocation," http://www.hpl.hp.com/research/tycoon/index.html, last visited: June 2017.

[152] B. N. Chun and D. E. Culler, "Market-based proportional resource sharing for clusters," EECS Department, University of California, Berkeley, Tech. Rep., 2000.

[153] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman, "Tycoon: An implementation of a distributed, market-based resource allocation system," *Multiagent Grid Systems*, vol. 1, no. 3, pp. 169–182, August 2005.

[154] F. Wu and L. Zhang, "Proportional response dynamics leads to market equilibrium," in *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, 2007, pp. 354–363.

[155] L. Zhang, "Proportional response dynamics in the Fisher market," *Theoretical Computer Science*, vol. 412, no. 24, pp. 2691–2698, 2011.

[156] K. J. Arrow and G. Debreu, "Existence of an equilibrium for a competitive economy," *Econometrica: Journal of the Econometric Society*, pp. 265–290, 1954.

[157] L. Zhang, "Proportional response dynamics in the Fisher market," *Theoretical Computer Science*, vol. 412, no. 24, pp. 2691–2698, 2011.

[158] A. Snavely and D. M. Tullsen, "Symbiotic job scheduling for a simultaneous multithreaded processor," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000, pp. 234–244.

[159] S. Boyd and L. Vandenberghe, *Convex optimization.* Cambridge university press, 2004.

[160] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu, "VMware distributed resource management: Design, implementation, and lessons learned," *VMware Technical Journal*, vol. 1, no. 1, pp. 45–64, 2012.

[161] Q. Llull, S. Fan, S. M. Zahedi, and B. C. Lee, "Cooper: Task colocation with cooperative games," in *Proceedings of the 23rd IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 421–432.

[162] O. Agmon Ben-Yehuda, E. Posener, M. Ben-Yehuda, A. Schuster, and A. Mu'alem, "Ginseng: Market-driven memory allocation," in *Proceedings of*

the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, 2014.

[163] Q. Pu, H. Li, M. Zaharia, A. Ghodsi, and I. Stoica, "FairRide: Near-optimal, fair cache sharing," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[164] L. B. N. Laboratory, "National energy research scientific computing center," https://www.nersc.gov, 2017.

[165] "Sharcnet," https://www.sharcnet.ca, Accessed Jan 2018.

[166] "Fair scheduler," https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html, Accessed Jan 2018.

[167] "Capacity scheduler," https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html, Accessed Jan 2018.

[168] "Spark dynamic allocation," https://spark.apache.org/docs/latest/job-scheduling.html, Accessed Jan 2018.

[169] G. Birkhoff, "Tres observaciones sobre el algebra lineal," *Univ. Nac. Tucumán Rev. Ser. A*, vol. 5, pp. 147–151, 1946.

[170] A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Choosy: Max-min fair sharing for datacenter jobs with constraints," in *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys)*, 2013, pp. 365–378.

[171] A. Shieh, S. Kandula, A. G. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, vol. 11, 2011, pp. 23–23.

[172] M. Aleksandrov, H. Aziz, S. Gaspers, and T. Walsh, "Online fair division: Analysing a food bank problem," in *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015, pp. 2540–2546.

[173] A. Gulati, G. Shanmuganathan, X. Zhang, and P. J. Varman, "Demand based hierarchical qos using storage resource pools," in *USENIX Annual Technical Conference*, 2012, pp. 1–13.

[174] "More Google cluster data," Google research blog, Nov 2011, posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

[175] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format + Schema," Google Inc., Mountain View, CA, USA, Technical Report, Nov 2011, revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data.

[176] A. Gorokh, S. Banerjee, and K. Iyer, "From monetary to non-monetary mechanism design via artificial currencies (Extended Abstract)," in *Proceedings of the 18th ACM Conference on Economics and Computation (EC)*, 2017.

[177] M. Guo, V. Conitzer, and D. M. Reeves, "Competitive repeated allocation without payments," in *Proceedings of the 5th International Workshop on Internet and Network Economics (WINE)*, 2009, pp. 244–255.

[178] S. Athey and K. Bagwell, "Optimal collusion with private information," *RAND Journal of Economics*, vol. 32, no. 3, pp. 428–465, 2001.

[179] A. Abdulkadiroğlu and K. Bagwell, "Trust, reciprocity, and favors in cooperative relationships," *American Economic Journal: Microeconomics*, vol. 5, no. 2, pp. 213–259, 2013.

[180] M. Aleksandrov and T. Walsh, "Pure nash equilibria in online fair division," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, 2017, pp. 42–48.

[181] E. Friedman, C.-A. Psomas, and S. Vardi, "Controlled dynamic fair division," in *Proceedings of the 2017 ACM Conference on Economics and Computation (EC)*, 2017, pp. 461–478.

[182] T. Walsh, "Online cake cutting," in *Proceedings of the 2nd International Conference on Algorithmic Decision Theory (ADT)*, 2011, pp. 292–305.

[183] I. Kash, A. D. Procaccia, and N. Shah, "No agent left behind: Dynamic fair division of multiple resources," *Journal of Artificial Intelligence Research*, vol. 51, pp. 579–603, 2014.

[184] K. J. Duda and D. R. Cheriton, "Borrowed-virtual-time (BVT) scheduling: Supporting latency-sensitive threads in a general-purpose scheduler," in *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles (SOSP)*, 1999, pp. 261–276.

[185] S. Tang, B.-S. Lee, B. He, and H. Liu, "Long-term resource fairness: Towards economic fairness on pay-as-you-use computing systems," in *Proceedings of*

the 28th ACM International Conference on Supercomputing (ICS), 2014, pp. 251–260.

[186] T. Sandholm and K. Lai, “Dynamic proportional share scheduling in Hadoop,” in *Workshop on Job Scheduling Strategies for Parallel Processing.* Springer, 2010, pp. 110–131.

Biography

Seyed Majid Zahedi was born in April 4th, 1987 in Tehran, Iran. He is a Ph.D. candidate in Computer Science at Duke University, advised by Prof. Benjamin C. Lee. Prior to his Ph.D., he obtained his M.S. and B.S. degrees from University of Tehran, Iran. His research focuses on the intersection of computer architecture, computer systems, and economic game theory. His work has been acclaimed by the computer architecture community. His paper on the Amdahl bidding mechanism received the Best Paper Award at HPCA'18, his paper on computational sprinting received the Best Paper Award at ASPLOS'16, was selected as a CACM Research Highlight, and was distinguished as one of the IEEE Micro Top Picks Honorable Mentions, and his paper on multi-resource allocation was recognized as one of the IEEE Micro Top Picks from Computer Architecture Conferences in 2014.