Managing Heterogeneous Datacenters with Tokens

SEYED MAJID ZAHEDI, Duke University
SONGCHUN FAN, Google, Duke University
BENJAMIN C. LEE, Duke University

Ensuring fairness in a system with scarce, 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, which frames the allocation problem as a repeated game. At each round, users request big processors and spend a token if their request is granted. We analyze the game and optimize users' strategies to produce an equilibrium. In equilibrium, allocations balance performance and fairness. Our mechanism outperforms classical, fair mechanisms by 1.7×, on average, in performance gains, and is competitive with a performance maximizing mechanism.

CCS Concepts: • **Software and its engineering → Power management**; • **Information systems → Data centers**; • **Social and professional topics → Pricing and resource allocation**; • **Theory of computation → Algorithmic mechanism design**; • **Computing methodologies →** *Modeling methodologies*;

Additional Key Words and Phrases: Modeling, power, allocation and scheduling, resource management, optimization, equilibrium

ACM Reference format:
Seyed Majid Zahedi, Songchun Fan, and Benjamin C. Lee. 2018. Managing Heterogeneous Datacenters with Tokens. *ACM Trans. Archit. Code Optim.* 15, 2, Article 18 (May 2018), 23 pages.
https://doi.org/10.1145/3191821

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 [35, 37], instruction throughput in chip multiprocessors [11], or service quality in datacenters [8, 21, 22, 46].

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, then he or 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

This work was supported by NSF grants CCF-1149252 (CAREER), CCF-1337215 (XPS-CLCCA), SHF-1527610, and AF-1408784. This work was also supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of these sponsors.
Authors' addresses: S. M. Zahedi, Duke University, 308 Research Drive, Durham, NC, 27708, USA; email: zahedi@cs.duke.edu; S. Fan, Google, Duke University, 308 Research Drive, Durham, NC, 27708, USA; email: songchun.fan@duke.edu; B. C. Lee, Duke University, Hudson Hall, Research Dr. Durham, NC, 27708, USA; email: benjamin.c.lee@duke.edu.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 ACM 1544-3566/2018/05-ART18 $15.00
https://doi.org/10.1145/3191821

(EF) [62]. 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 [10, 19, 24, 30, 69]. 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 article, 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 [63], 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 [17, 69].

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 2 and 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)
- **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 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., 1.7× better than round-robin) and performs comparably to performance-maximizing policies. (Sections 6 and 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.

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 [62], 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. [19]. Zahedi et al. propose Resource Elasticity Fairness when allocating cache capacity and memory bandwidth in chip multiprocessors [69]. These algorithms guarantee SI and EF for very different performance models, pursuing fairness in space.

Repeated games. In this article, 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 [63], that rigidly enforce SI and EF in every round.

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 [29, 47, 50], adaptive microarchitectures [27, 33, 44], and power budgets [12, 16, 54].

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.

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, then each agent who prefers a boost receives one. If more than n agents signal Y, then 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 (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 $\sum_{t>t'} f(t)$ percentage 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 $P_R = n/(m - n)$. The game does not use fractional tokens and redistributes tokens probabilistically, which is fair in expectation.

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 runtime. 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, and
- 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.

[1]Users could start with different token holdings if they have different weights (i.e., priorities)—see Section 9.

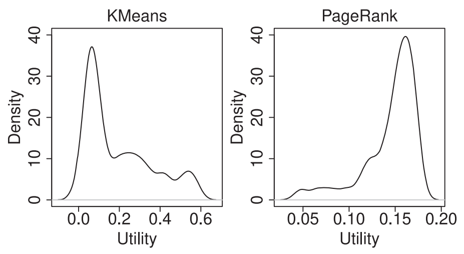

Fig. 1. Probability density functions on utility (i.e., throughput gains) from power boosts.

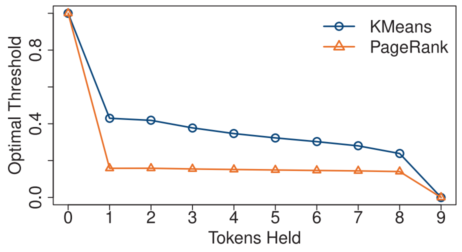

Fig. 2. Signaling thresholds. Agents with more tokens lower thresholds and spend tokens more freely. KMeans agents have higher utilities and thresholds when choosing rounds in which to signal.

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.

3.3 Game Dynamics

We optimize each agent's strategy in Section 4 but first present an extended case study that illustrates game play. See Section 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 [13, 70] 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 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 2 presents representative thresholds, which 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.

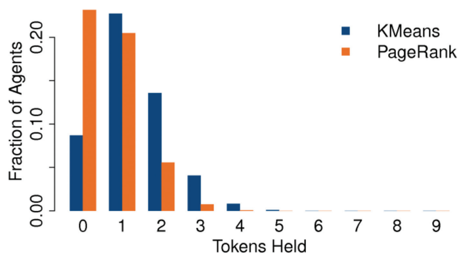

Fig. 3. Tokens. KMeans agents have higher thresholds, signal less for boosts, receive them less frequently, and hold more tokens.

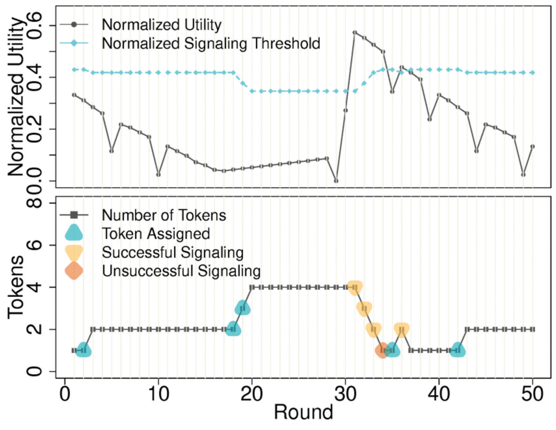

Fig. 4. Snapshot of system dynamics.

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

Game play. Figure 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 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 [2–4, 23, 28]. 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 $P_B = P'_B$, then 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.

4.1 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 $\neg Y$,

$$V(t,u) = \max(V_Y(t,u), V_{\neg Y}(t,u)). \tag{1}$$

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,

$$\begin{aligned} V_Y(t,u) = &\, P_B(t)(u + \gamma V(t-1)) \\ &+ (1 - P_B(t))(P_R \gamma V(t+1) + (1 - P_R)\gamma V(t)). \end{aligned} \tag{2}$$

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,

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

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,

$$V_{\neg Y}(t,u) = \gamma(P_R V(t+1) + (1 - P_R)V(t)). \tag{3}$$

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 (2) and (3), agents should signal if boosted utility u exceeds threshold u_{thr},

$$u > \underbrace{\gamma(P_R V(t+1) + (1 - P_R)V(t) - V(t-1))}_{u_{\text{thr}}(t)}. \tag{4}$$

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

$$P_Y(t) = \Pr(u \geq u_{\text{thr}}(t)) = \int_{u \geq u_{\text{thr}}(t)} h(u)du. \tag{5}$$

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.

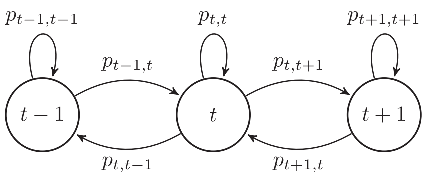
$$\begin{tikzcd} t-1 \arrow[loop above, "p_{t-1,t-1}"] \arrow[r, bend left, "p_{t-1,t}"] & t \arrow[loop above, "p_{t,t}"] \arrow[l, bend left, "p_{t,t-1}"] \arrow[r, bend left, "p_{t,t+1}"] & t+1 \arrow[loop above, "p_{t+1,t+1}"] \arrow[l, bend left, "p_{t+1,t}"] \end{tikzcd}$$

Fig. 5. Token exchange.

Agents implement their signaling strategies with offline analysis and online comparisons, sketched here and detailed in Section 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.

4.2 Assess Token Distribution

Agents exchange tokens as the game allocates power boosts. Figure 5 presents a Markov chain that specifies possible token holdings and transitions between them. An agent with i tokens has j tokens in the next round with probability $p_{i,j}$,

$$\begin{aligned} p_{t,t-1} &= P_Y(t)P_B(t) \\ p_{t,t} &= (1-P_R)(1-P_Y(t)P_B(t)) \\ p_{t,t+1} &= P_R(1-P_Y(t)P_B(t)). \end{aligned} \tag{6}$$

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 $t \in (0, t_{\max})$,

$$f(t) = f(t+1)p_{t+1,t} + f(t)p_{t,t} + f(t-1)p_{t-1,t}. \tag{7}$$

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.

4.3 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,

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

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

$$P_B(t) = \begin{cases} 0 & \text{if } mG(t+1) \geq n, \\ 1 & \text{if } mG(t) < n, \\ \frac{n-mG(t+1)}{mg(t)} & \text{otherwise.} \end{cases} \tag{8}$$

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 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 (1) through (3).
- f is token distribution that satisfies consistency conditions in Equations (6) and (7).
- P_B is signaling strength that satisfies consistency conditions in Equation (8).

Algorithm 1 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.

ALGORITHM 1: 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 (1)–(4) given $P_B$
    $P_Y \leftarrow$ Equation (5) given $h(u)$, $u_{\text{thr}}$
    $f(t_{ini}) \leftarrow 1$
    while $f$ not converged do
        $f \leftarrow$ Equations (6)–(7) given $P_Y$, $P_B$, $f$
        $P_B \leftarrow$ Equation (8) given $f$, $P_Y$
    end
end
```

To prove that the mean field equilibrium exists and Algorithm 1 finds the equilibrium, we need to use a fixed point theorem. This is out of the scope of this article 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.

5 GAME ARCHITECTURE

Figure 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.

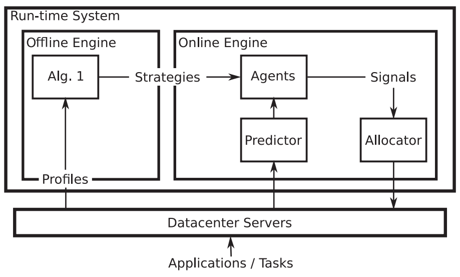

Fig. 6. Game architecture.

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 1 to optimize threshold strategies based on utility distributions. We implement the algorithm in R. Algorithm 1 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 [41]. 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 $(1-\gamma)^{-1}$.

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 $O(m \log m)$ time. A simple approach sorts agents in every round, which takes less than 1ms for 1000 agents and less than 4ms for 10,000 agents.

The allocator implements power boosts with Intel RAPL [1]. 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.

Table 1. Spark Workloads

ID	Applications	Dataset	Data Size
1	`Correlation`	kdda2010 [59]	2.5G
2	`FP Growth`	Webdocs [43]	1.5G
3	`KMeans`	uscensus1990 [39]	327M
4	`LinearRegression`	kddb2010 [59]	4.8G
5	`ALS`	movielens2015 [25]	325M
6	`SVM`	kdda2010	2.5G
7	`Pagerank`	wdc2012 [48]	5.3G
8	`ConnectedComponents`	wdc2012	5.3G
9	`TriangleCounting`	wdc2012	5.3G

Table 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

6 EXPERIMENTAL METHODOLOGY

Spark workloads. We evaluate task-parallel datacenter workloads from Apache Spark [68] (see Table 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 [1]. We set power limits for sockets by writing to the `MSR_PKG_POWER_LIMIT` register. We trace utility from power boosts by measuring 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 2 summarizes the simulation of 1,000 agents that run varied workloads for 3,000 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 3. The datacenter's power delivery unit (PDU) supplies 35KW of power. Power is shared by 1,000-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 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.

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

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:

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

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.

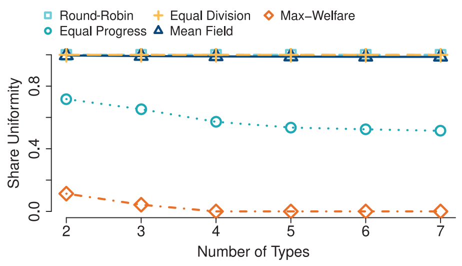

Fig. 7. Share uniformity. M-F, E-D, and R-R achieve uniform shares; M-W and E-P violate SI_R.

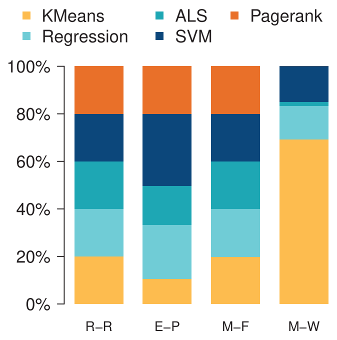

Fig. 8. Sharing incentives. Time in boosted rounds.

Figure 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 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 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.

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

[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.

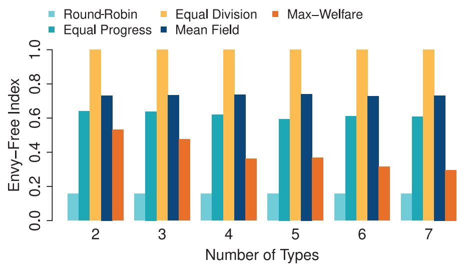

Fig. 9. Envy-free Index. E-D achieves envy-freeness. M-F has high envy-free index. R-R, E-P, and M-W ignore envy.

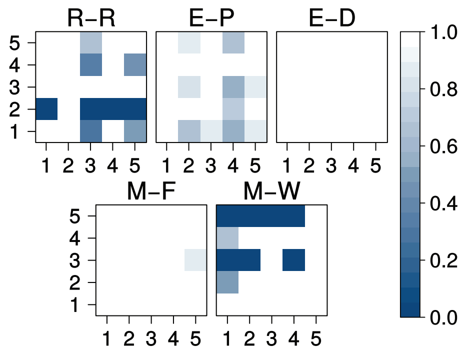

Fig. 10. Envy-free index. Darker colors mean greater envy. Five types include (1) KMeans, (2) Regression, (3) ALS, (4) SVM, (5) Pagerank.

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 9 evaluates EF indices for diverse agent populations. Figure 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 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, then they did not need boosts and are not envious despite receiving nominal power budgets. If these agents signaled unsuccessfully, then they must have already spent tokens for boosts in prior rounds. Failed signals induce little envy, because swapping

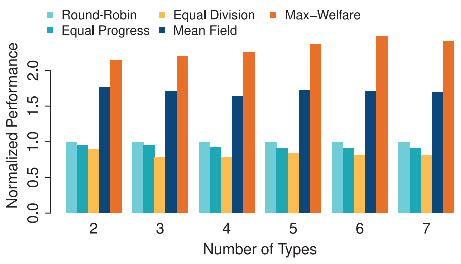

Fig. 11. Performance for 3 to 12 types. M-F outperforms R-R, E-D, and E-P and is comparable with M-W.

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 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.

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, then 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 11 presents performance for diverse agent populations. M-F outperforms R-R by 1.7×, 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 2×, on average. E-D divides 35KW power budget among 1,000 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 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.

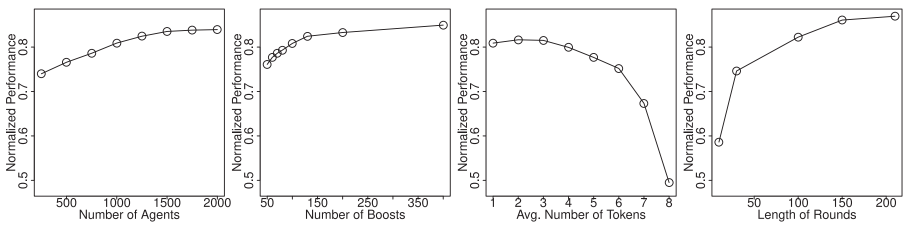

Fig. 12. Sensitivity: (a) number of agents, (b) number of power boosts, (c) initial number of tokens.

7.4 Sensitivity to Game Parameters

Figure 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 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.

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, then profiles will less accurately capture utility and strategies will fall short of optimal responses to the competition from other jobs.

Figure 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.

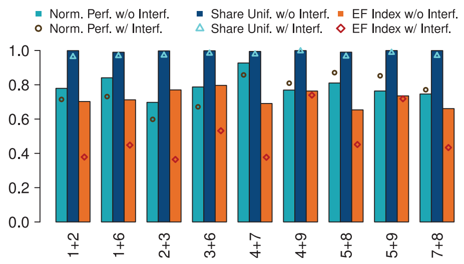

Fig. 13. Sensitivity to interference. Performance, share uniformity, and envy-free index are affected due to interference.

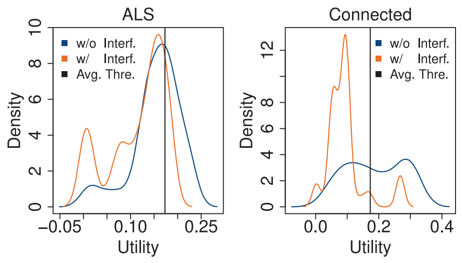

Fig. 14. Utility distribution with and without interference between ALS and Connected. Threshold is optimized assuming no interference and averaged across token holdings.

Figure 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 vs. 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 [26, 45, 57, 65]. Finally, agents could continuously update their utility profiles and re-optimize their thresholds.

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 [19]. Resource elasticity fairness makes similar assurances when allocating cache capacity and memory bandwidth [69]. Wang et al. present a market with dynamic price discovery, which balances throughput and fairness [66]. These mechanisms guarantee fairness in space when there are many more resources than users.

For fairness in time, Craeynest et al. [60] 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. [20] 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 [52]. Ghodsi et al. [18] 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 [9, 53].

Scrip systems. Friedman et al. [15] 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. [31] extend the scrip system for heterogeneous users and [32] extends this model by studying the effect of collusion.

Buttyan et al. [6] propose a scrip system to stimulate cooperation in ad hoc mobile networks. Considering a similar setting, Xu et al. [67] design a token system to incentivize self-interested users to relay other nodesâĂŹ traffic in autonomic wireless relay networks. Shen et al. [58] 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. [5] 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 [7, 34, 44, 61]. Guevara et al. allocate heterogeneous processors using a market to ensure service quality [21, 22]. Unlike previous works, we focus on game-theoretic notion of fairness while maximizing overall performance.

Fan et al. [12] study a computational sprinting game in which multi-core chips share a power supply and sprint independently. Similar to our article, Fan et al. [12] use dynamic programming to find mean field equilibrium strategies for power boosts. However, our article differs from Fan et al. [12], 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 our 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 [38, 56]. Femal et al. [14] 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 [40, 55, 64]. Co-Con [36] uses a power control loop and a performance control loop to make adjustments on power and performance at the cluster level. PEGASUS [42] uses a feedback-based controller to dynamically assign power caps to the most latency critical workloads. Finally, VPM Tokens [51] manages power from virtual machines' perspective while considering global performance.

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 [49]. 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.

REFERENCES

[1] Intel 64 and IA-32 Architectures Software Developer's Manual. Retrieved April 4, 2018, from 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.

[2] Sachin Adlakha and Ramesh Johari. 2013. Mean field equilibrium in dynamic games with strategic complementarities. *Operat. Res.* 61, 4, 971–989.

[3] Sachin Adlakha, Ramesh Johari, and Gabriel Y. Weintraub. 2015. Equilibria of dynamic games with many players: Existence, approximation, and market structure. *Journal of Economic Theory* 156 (March 2015), 269–316.

[4] Sachin Adlakha, Ramesh Johari, Gabriel Y. Weintraub, and Andrea Goldsmith. 2010. On oblivious equilibrium in large population stochastic games. In *Proceedings of the 49th IEEE Conference on Decision and Control (CDC'10).* IEEE, Los Alamitos, CA, 3117–3124.

[5] Matthew Andrews, Lijun Qian, and Alexander Stolyar. 2005. 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'05)*, Vol. 4. IEEE, Los Alamitos, CA, 2415–2424.

[6] Levente Buttyan and Jean-Pierre Hubaux. 2001. *Nuglets: A Virtual Currency to Stimulate Cooperation in Self-Organized Mobile Ad Hoc Networks.* Technical Report. Swiss Federal Institute of Technology.

[7] Jian Chen and Lizy K. John. 2009. Efficient program scheduling for heterogeneous multi-core processors. In *Proceedings of the 46th Annual Design Automation Conference (DAC'09).* ACM, New York, NY, 927–930.

[8] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'13).* ACM, New York, NY, 77–88.

[9] Alan Demers, Srinivasan Keshav, and Scott Shenker. 1989. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM Comput. Commun. Rev.* 19, 1–12.

[10] Danny Dolev, Dror G. Feitelson, Joseph Y. Halpern, Raz Kupferman, and Nathan Linial. 2012. No justified complaints: On fair sharing of multiple resources. In *Proceedings of the Innovations in Theoretical Computer Science Conference.* ACM, New York, NY, 68–75.

[11] Stijn Eyerman and Lieven Eeckhout. 2008. System-level performance metrics for multiprogram workloads. *IEEE Micro* 28, 3, 42–53.

[12] Songchun. Fan, Seyed Majid Zahedi, and Benjamin C. Lee. 2016. The computational sprinting game. In *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'16)*. ACM, New York, NY, 561–575.

[13] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. 2007. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*. ACM, New York, NY, 13–23.

[14] Mark E. Femal and Vincent W. Freeh. 2005. Boosting data center performance through non-uniform power allocation. In *Proceedings of the 2nd International Conference on Automatic Computing (ICAC'05)*. IEEE, Los Alamitos, CA, 250–261.

[15] Eric J. Friedman, Joseph Y. Halpern, and Ian Kash. 2006. Efficiency and Nash equilibria in a scrip system for P2P networks. In *Proceedings of the 7th ACM Conference on Electronic Commerce*. ACM, New York, NY, 140–149.

[16] Xing Fu, Xiaorui Wang, and Charles Lefurgy. 2011. How much power oversubscription is safe and allowed in data centers. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC'11)*. ACM, New York, NY, 21–30.

[17] Ron Gabor, Shlomo Weiss, and Avi Mendelson. 2006. Fairness and throughput in switch on event multithreading. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE, Los Alamitos, CA, 149–160.

[18] Ali Ghodsi, Vyas Sekar, Matei Zaharia, and Ion Stoica. 2012. Multi-resource fair queueing for packet processing. *ACM SIGCOMM Comput. Commun. Rev.* 42, 4, 1–12.

[19] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI'11)*. 323–336.

[20] Artur Gorokh, Siddhartha Banerjee, and Krishnamurthy Iyer. 2016. Near-efficient allocation using artificial currency in repeated settings (extended abstract). In *Proceedings of the 12th International Conference on Web and Internet Economics (WINE'16)*.

[21] Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. 2013. Navigating heterogeneous processors with market mechanisms. In *Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture (HPCA'13)*. IEEE, Los Alamitos, CA, 95–106.

[22] Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. 2014. Strategies for anticipating risk in heterogeneous system design. In *Proceedings of the 20th IEEE International Symposium on High Performance Computer Architecture (HPCA'14)*. IEEE, Los Alamitos, CA, 154–164.

[23] Ramakrishna Gummadi, Ramesh Johari, and Jia Yuan Yu. 2012. Mean field equilibria of multiarmed bandit games. In *Proceedings of the 13th ACM Conference on Electronic Commerce (EC'12)*. ACM, New York, NY, 655.

[24] Avital Gutman and Noam Nisan. 2012. Fair allocation without trade. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*. 719–728.

[25] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens datasets: History and context. *ACM Trans. Interact. Intell. Syst.* 5, 4, 19.

[26] Derek R. Hower, Harold W. Cain, and Carl A. Waldspurger. 2017. PABST: Proportional allocation of bandwidth at the source and target. In *Proceedings of the 23rd IEEE International Symposium on High Performance Computer Architecture (HPCA'17)*.

[27] Engin Ipek, Meyrem Kirman, Nevin Kirman, and Jose F. Martinez. 2007. Core fusion: Accommodating software diversity in chip multiprocessors. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*. ACM, New York, NY, 186–197.

[28] Krishnamurthy Iyer, Ramesh Johari, and Mukund Sundararajan. 2011. Mean field equilibria of dynamic auctions with learning. *ACM SIGecom Exch.* 10, 3, 10–14.

[29] Vijay Janapa Reddi, Benjamin C. Lee, Trishul Chilimbi, and Kushagra Vaid. 2010. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10)*.

[30] Carlee Joe-Wong, Soumya Sen, Tian Lan, and Mung Chiang. 2012. Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework. In *Proceedings of IEEE International Conference on Computer Communications (INFOCOM'12)*. IEEE, Los Alamitos, CA.

[31] Ian A. Kash, Eric J. Friedman, and Joseph Y. Halpern. 2007. Optimizing scrip systems: Efficiency, crashes, hoarders, and altruists. In *Proceedings of the 8th ACM Conference on Electronic Commerce*. ACM, New York, NY, 305–315.

[32] Ian A. Kash, Eric J. Friedman, and Joseph Y. Halpern. 2009. Manipulating scrip systems: Sybils and collusion. In *Auctions, Market Mechanisms and Their Applications*. Springer, 13–24.

[33] K. Khubaib, M. Aater Suleman, Milad Hashemi, Chris Wilkerson, and Yale N. Patt. 2012. 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'12)*. IEEE, Los Alamitos, CA, 305–316.

[34] David Koufaty, Dheeraj Reddy, and Scott Hahn. 2010. Bias scheduling in heterogeneous multi-core architectures. In *Proceedings of the 5th European Conference on Computer Systems*. ACM, New York, NY, 125–138.

[35] Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, and Dean M. Tullsen. 2003. 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'03)*. IEEE, Los Alamitos, CA, 81–92.

[36] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. 2009. Power and performance management of virtualized computing environments via lookahead control. *Cluster Comput.* 12, 1, 1–15.

[37] Benjamin C. Lee and David M. Brooks. 2007. Illustrative design space studies with microarchitectural regression models. In *Proceedings of the 13th IEEE International Symposium on High Performance Computer Architecture (HPCA'07)*. IEEE, Los Alamitos, CA, 340–351.

[38] C. Lefurgy, X. Wang, and M. Ware. 2007. Server-level power control. In *Proceedings of the 4th International Conference on Autonomic Computing (ICAC'07)*. IEEE, Los Alamitos, CA.

[39] M. Lichman. 2013. UCI Machine Learning Repository. Retrieved April 5, 2018, from http://archive.ics.uci.edu/ml.

[40] Harold Lim, Aman Kansal, and Jie Liu. 2011. Power budgeting for virtualized data centers. In *Proceedings of the 2011 USENIX Annual Technical Conference*.

[41] Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. 1995. On the complexity of solving Markov decision problems. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*. 394–402.

[42] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. 2014. Towards energy proportionality for large-scale latency-critical workloads. In *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA'14)*. IEEE, Los Alamitos, CA, 301–312.

[43] Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fabrizio Silvestri. 2004. WebDocs: A real-life huge transactional dataset. In *Proceedings of the Workshop on Frequent Itemset Mining Implementations*.

[44] Andrew Lukefahr, Shruti Padmanabha, Reetuparna Das, Faissal M. Sleiman, Ronald Dreslinski, Thomas F. Wenisch, and Scott Mahlke. 2012. Composite cores: Pushing heterogeneity into a core. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'12)*. IEEE, Los Alamitos, CA, 317–328.

[45] R. Manikantan, K. Rajan, and R. Govindarajan. 2012. Probabilistic shared cache management (PriSM). In *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA'12)*. 428–439.

[46] Jason Mars and Lingjia Tang. 2013. Whare-map: Heterogeneity in homogeneous warehouse-scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*. ACM, New York, NY, 619–630.

[47] Jason Mars and Lingjia Tang. 2013. Whare-map: Heterogeneity in homogeneous warehouse-scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*.

[48] Robert Meusel, Sebastiano Vigna, Oliver Lehmberg, and Christian Bizer. 2012. Web Data Commons—Hyperlink Graphs. Retrieved December 29, 2016, from http://webdatacommons.org/hyperlinkgraph/index.html.

[49] Nathaniel Morris, Siva Meenakshi Renganathan, Christopher Stewart, Robert Birke, and Lydia Chen. 2016. Sprint ability: How well does your software exploit bursts in processing capacity? In *Proceedings of the 13th IEEE International Conference on Autonomic Computing (ICAC'16)*. IEEE, Los Alamitos, CA, 173–178.

[50] Ripal Nathuji, Canturk Isci, and Eugene Gorbatov. 2007. Exploiting platform heterogeneity for power efficient data centers. In *Proceedings of the 4th International Conference on Autonomic Computing (ICAC'07)*.

[51] Ripal Nathuji and Karsten Schwan. 2008. VPM tokens: Virtual machine-aware power budgeting in datacenters. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*. ACM, New York, NY, 119–128.

[52] Kyle J. Nesbit, Nidhi Aggarwal, James Laudon, and James E. Smith. 2006. Fair queuing memory systems. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE, Los Alamitos, CA, 208–222.

[53] Abhay K. Parekh and Robert G. Gallager. 1993. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Trans. Netw.* 1, 3, 344–357.

[54] Arun Raghavan, Yixin Luo, Anuj Chandawalla, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M. K. Martin. 2012. Computational sprinting. In *Proceedings of the 18th IEEE International Symposium on High Performance Computer Architecture (HPCA'12)*. IEEE, Los Alamitos, CA, 1–12.

[55] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. 2008. 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'08)*. ACM, New York, NY, 48–59.

[56] Parthasarathy Ranganathan, Phil Leech, David Irwin, and Jeffrey Chase. 2006. Ensemble-level power management for dense blade servers.In *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA'06)*. 66–77.

[57] Daniel Sanchez and Christos Kozyrakis. 2011. Vantage: Scalable and efficient fine-grain cache partitioning. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA'11)*. 57–68.

[58] Cong Shen, Jie Xu, and Mihaela van der Schaar. 2015. Silence is gold: Strategic interference mitigation using tokens in heterogeneous small cell networks. *IEEE J. Select. Areas Commun.* 33, 6, 1097–1111.

[59] J. Stamper, A. Niculescu-Mizil, S. Ritter, G. J. Gordon, and K. R. Koedinger. 2010. Algebra I 2006-2007. Challenge data set from KDD Cup 2010 Educational Data Mining Challenge. Retrieved April 5, 2018, from http://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp.

[60] Kenzo Van Craeynest, Shoaib Akram, Wim Heirman, Aamer Jaleel, and Lieven Eeckhout. 2013. Fairness-aware scheduling on single-ISA heterogeneous multi-cores. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT'13)*. IEEE, Los Alamitos, CA, 177–187.

[61] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, and Joel Emer. 2012. Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA'12)*. IEEE, Los Alamitos, CA.

[62] Hal Varian. 1974. Equity, envy, and efficiency. *J. Econ. Theory* 9, 1, 63–91.

[63] Carl A. Waldspurger and William E. Weihl. 1994. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI'94)*.

[64] X. Wang, M. Chen, C. Lefurgy, and T. W. Keller. 2009. SHIP: Scalable hierarchical power control for large-scale data centers. In *Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques (PACT'09)*. 91–100.

[65] Xiaodong Wang, Shuang Chen, Jeff Setter, and José F. Martínez. 2017. 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'17)*.

[66] Xiaodong Wang and José F. Martínez. 2015. XChange: Scalable dynamic multi-resource allocation in multicore architectures. In *Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture (HPCA'15)*.

[67] Jie Xu and Mihaela Van Der Schaar. 2013. Token system design for autonomic wireless relay networks. *IEEE Trans. Commun.* 61, 7, 2924–2935.

[68] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10)*, Vol. 10.

[69] Seyed Majid Zahedi and Benjamin C. Lee. 2014. 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'14)*. ACM, New York, NY, 145–160.

[70] Wenli Zheng and Xiaorui Wang. 2015. Data center sprinting: Enabling computational sprinting at the data center level. In *Proceedings of the 35th International Conference on Distributed Computing Systems (ICDCS'15)*. IEEE, Los Alamitos, CA, 175–184.

Received September 2017; revised January 2018; accepted February 2018