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

Seyed Majid Zahedi,*† Qiuyun Llull,*†‡ Benjamin C. Lee†
†*Duke University, Durham, NC, USA*
‡*VMware Inc., Palo Alto, CA, USA*
seyedmajid.zahedi@duke.edu, qllull@vmware.com, benjamin.c.lee@duke.edu

Abstract—**We present a processor allocation framework that uses Amdahl's Law to model parallel performance and a market mechanism to allocate cores. First, we propose the Amdahl utility function and demonstrate its accuracy when modeling performance from processor core allocations. Second, we design a market based on Amdahl utility that optimizes users' bids for processors based on workload parallelizability. The framework uses entitlements to guarantee fairness yet outperforms existing proportional share algorithms.**

Keywords-**Proportional Sharing, Processor Allocation, Amdahl's Law, Karp-Flatt Metric, Market Mechanisms**

I. 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 [1], [2], [3]. 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 [4], [5]. With priorities, allocations depend on users' computation and relative importance, exposing users to interference and non-deterministic performance [6], [2]. Finally, with entitlements, each user is guaranteed a minimum allocation and under-utilized resources are redistributed [6], [1], [7], [3]. 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 [6], [7]. 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 [8], [9]. 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 [10], [11], [12], [13].

In this paper, we co-design a utility function and market mechanism for processor allocation (§II). We estimate the utility function's parameter, the workload's parallelizable fraction, by inverting Amdahl's Law (§III–§IV). We derive a procedure for calculating bids and allocations that produce a market equilibrium (§V). Finally, we find that equilibrium allocations satisfy entitlements and perform well. (§VI).

*Seyed Majid Zahedi and Qiuyun Llull contributed equally to this work. Qiuyun contributed during her doctoral studies at Duke and is currently employed by VMware.

II. Motivation and Overview

A. *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 non-determinism in performance from the Unix priority scheduler [6]. Kay and Lauder define fairness in terms of users rather than processes, which mitigates strategic behavior during heavy system activity [1]. Waldspurger and Weihl propose lottery scheduling, which allocates resources probabilistically based on users' holdings of a virtual currency [7]. 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 [1], [14], [15]. 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 [16]. Microsoft uses tokens, a form of lottery scheduling, to specify and enforce shares [3]. Google does not use entitlements and, consequently, suffers from the same challenges as other priority schedulers, which cannot guarantee performance isolation between users [2], [6].

B. *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.

Classical approaches, which enforce proportional shares on each server, violate 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:

$$\begin{aligned}\text{User 1} &\leftarrow (6_\text{A}, 4_\text{B}, 0_\text{C}),\\ \text{User 2} &\leftarrow (0_\text{A}, 4_\text{B}, 6_\text{C}),\\ \text{User 3} &\leftarrow (6_\text{A}, 4_\text{B}, 6_\text{C}).\end{aligned}$$

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.

$$\begin{aligned}\text{User 1} &\leftarrow (8_\text{A}, 4_\text{B}, 0_\text{C}),\\ \text{User 2} &\leftarrow (0_\text{A}, 4_\text{B}, 8_\text{C}),\\ \text{User 3} &\leftarrow (4_\text{A}, 4_\text{B}, 4_\text{C}).\end{aligned}$$

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.

C. *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 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 §II-B assumes that demand and utility could be represented with a single number, a popular approach in systems research [17], [18], [19]. 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.

D. 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 [8], [9]. 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:

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

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 paper, 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 [24], [25].

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 [26].

$$F = \left(1 - \frac{1}{s_x}\right)\left(1 - \frac{1}{x}\right)^{-1} \qquad (2)$$

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.

E. Mechanism Overview

We design a two-part mechanism for allocating datacenter processors given users' entitlements. First, the mechanism requires a utility function (§IV). 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 (§V). 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 (§VI).

III. 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 I summarizes our PARSEC [27] and Spark benchmarks [25] 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 II 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 [28]. 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.

IV. Performance Model

Equation (2) is an idealized estimate of a workload's parallel fraction. In principle, inherent properties of the algorithm or code determine parallelizability. The processor

Table I: Workloads and datasets

ID	Spark Name	Application	Dataset (Size)	ID	PARSEC Name	Application	Dataset
1	`Correlation`	Statistics	webspam2011 [20] (24GB)	13	`Blackscholes`	Finance	native
2	`Decision Tree`	Classifier	webspam2011 (24GB)	14	`Bodytrack`	Vision	native
3	`Fpgrowth`	Mining	wdc'12 [21] (1.4GB)	15	`Canneal`	Engineering	native
4	`Gradient Des.`	Classifier	webspam2011 (6GB)	16	`Dedup`	Storage	native
5	`Kmeans`	Clustering	uscensus [22] (327MB)	17	`Ferret`	Search	native
6	`Linear Reg.`	Classifier	webspam2011 (24GB)	18	`Raytrace`	Visualization	native
7	`Movie`	Recommender	movielens [23] (325MB)	19	`Streamcluster`	Data Mining	native
8	`Naive Bayes`	Classifier	webspam2011 (6GB)	20	`Swaptions`	Finance	native
9	`SVM`	Classifier	webspam2011 (24GB)	21	`Vips`	Media Proc.	native
10	`Page Rank`	Graph Proc.	wdc'12 [21] (5.3GB)	22	`X264`	Media Proc.	native
11	`Connected Cmp.`	Graph Proc.	wdc'12 (6GB)				
12	`Triangle Cnt.`	Graph Proc.	wdc'12 (5.3GB)				

Table II: 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

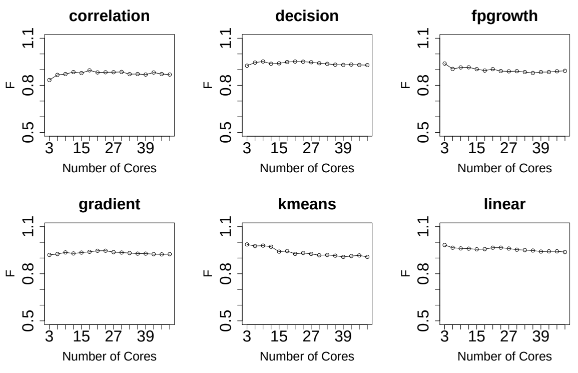

Figure 1: Calculated parallel fraction (F) for representative Spark workloads as processor count varies.

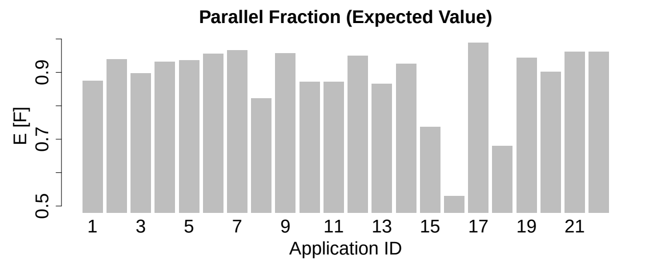

Figure 2: Expected parallel fraction, $\bar{F} = |x|^{-1}\sum_x F(x)$.

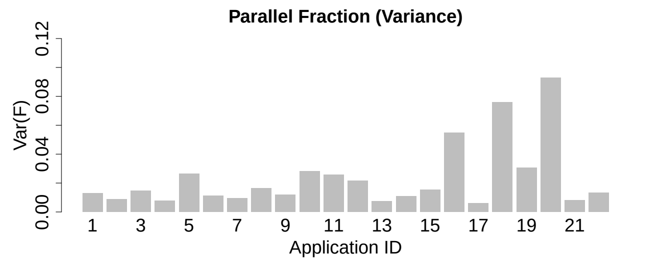

Figure 3: Variance in parallel fraction, $\text{Var}(F) = |x|^{-1}\sum_x (F(x) - \bar{F})^2$. Lower variance indicates a better fit with Amdahl's Law.

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.

$$F(x) = \left(1 - \frac{1}{s(x)}\right)\left(1 - \frac{1}{x}\right)^{-1} \tag{3}$$

In Equation (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 (2) is subtle but important. If the linearity assumption behind Amdahl's Law fails, the number of processors deployed to profile speedup will affect the estimated parallel fraction.

Figure 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 2 presents average estimates from varied processor counts. The parallel fraction ranges from 0.55 to 0.99 for Spark and PARSEC workloads. Figure 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`) [27] because adding processors increases overheads.

A. *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 4 shows how execution time scales linearly with dataset size when a representative workload, `correlation`, 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 [29].

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

B. *Predicting Parallel Performance*

Figure 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.

[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.

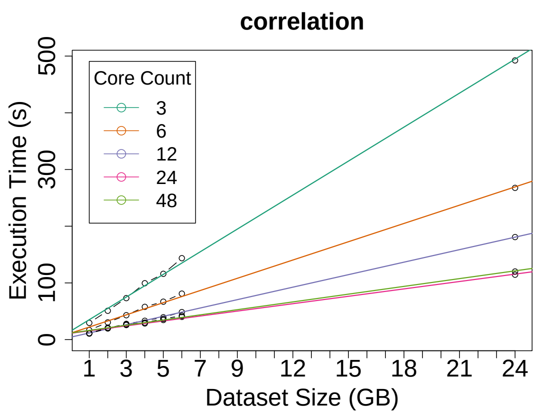

Figure 4: We sample the dataset size and measure performance for varied processor allocations. We fit linear models to estimate execution time from dataset size. Data shown for representative workload, `correlation`.

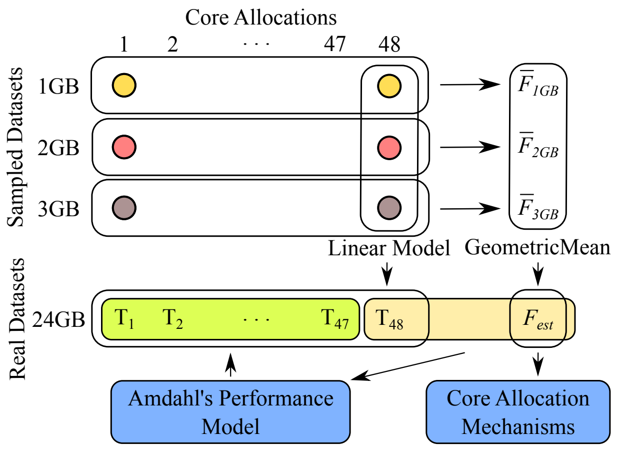

Figure 5: Use linear models to estimate effect of dataset size. Use Karp-Flatt analysis to estimate effect of processor count. Estimates can predict execution time or allocate processors.

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 accurate for varied parallel workloads—see §IV-C. 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 §V.

C. *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

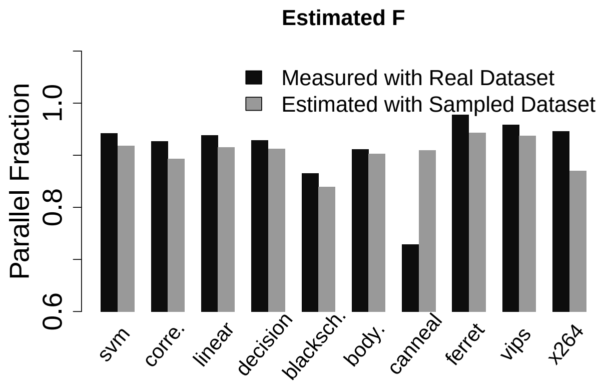

Figure 6: Accuracy of predicted parallel fraction F when using sampled datasets.

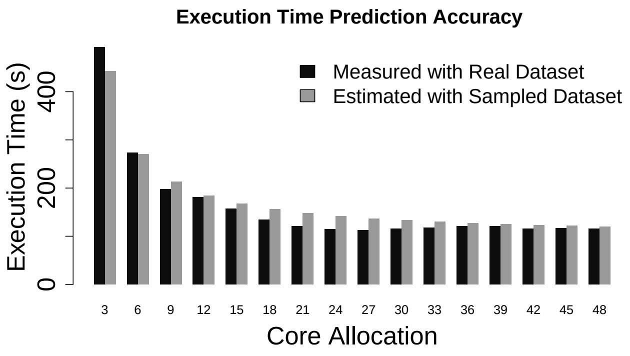

Figure 7: Accuracy of predicted execution time given varied processor counts. Data for `Decision Tree`.

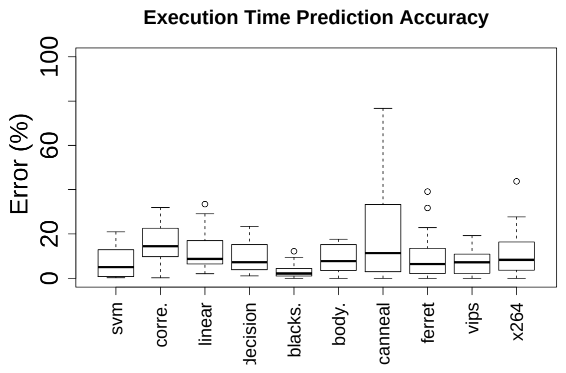

Figure 8: Accuracy of predicted execution time for varied applications. Boxplots show distribution of errors given varied processor allocations.

variety of workload inputs and processor allocations.

Parallel Fraction. Figure 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 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 over-estimate speedups from additional processors. The estimated parallel fraction is much larger than the one measured on the full dataset.

Execution Time. Figure 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 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 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.

V. 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. The problem is 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.

A. 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.

$$u_i(x_i) \;=\; \frac{\sum_{j=1}^{m} w_{ij} s_{ij}(x_{ij})}{\sum_{j=1}^{m} w_{ij}} \tag{4}$$

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.

B. 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.

$$\begin{aligned} \max \quad & u_i(x_i), \\ \text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} p_j \leq b_i. \end{aligned} \tag{5}$$

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]

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

[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.

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

C. 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) 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

$$\sum_j C_j p_j^* = B, \tag{6}$$

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 (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) for user i and price vector p^*. But since the equilibrium allocation x_i^* is the optimal solution,

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

D. Finding the Market Equilibrium

Several cluster managers have designed markets and used a particular algorithm—proportional response dynamics (PRD)—to find their equilibria[31], [32], [33]. PRD is an iterative algorithm that uses simple and proportional updates for bids and prices [34], [35]. It is decentralized, inexpensive, and avoids optimization.

[3]For strictly monotonic, continuous and non-satiable utility functions, optimal allocation exhausts user's budget [30].

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 [36]. And we know that PRD converges to the equilibrium when utility functions have constant elasticity of substitution (CES) [37].[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 §V-E. Our derivation re-writes 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) with a new variable $b_{ij} = x_{ij}p_j$, which is user i's bid for cores on server j.

$$\begin{aligned} \text{max.} \quad & u_i(x_i), && (7) \\ \text{s.t.} \quad & x_{ij} = b_{ij}/p_j, && \forall j, \\ & \sum_{j=1}^{m} b_{ij} \le b_i, & \\ & b_{ij} \ge 0, && \forall j. \end{aligned}$$

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.

$$p^*_j = \sum_i b^*_{ij}/C_j. \tag{8}$$

Finally, the second equilibrium condition states that, given prices p^*, allocations x^* and bids b^* should be the optimal solution for Optimization (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

$$\frac{{b^*_{ij}}^2}{{b^*_{ik}}^2} = \frac{f_{ij}\ p^*_j\ u^2_{ij}(x^*_{ij})}{f_{ik}\ p^*_k\ u^2_{ik}(x^*_{ik})}. \tag{9}$$

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

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.

E. Amdahl Bidding Procedure

Equations (8) and (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.

VI. Processor Allocation

We deploy workloads on physical servers (§III), profiling execution time on varied core allocations and datasets to fit Amdahl utility functions (§IV). Then, we construct a population of users that shares datacenter servers and run the market to allocate cores (§V). 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 I. 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 §V. It also corresponds to the weighted speedup metric, which is used to study multi-threaded and multi-core systems [38], [39].

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.

$$\text{SysProgress} = (1/B) \sum_i b_i \; \text{UserProgress}_i \qquad (10)$$

A. 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 (10). This policy favors users with larger budgets and entitlements when pursuing performance.

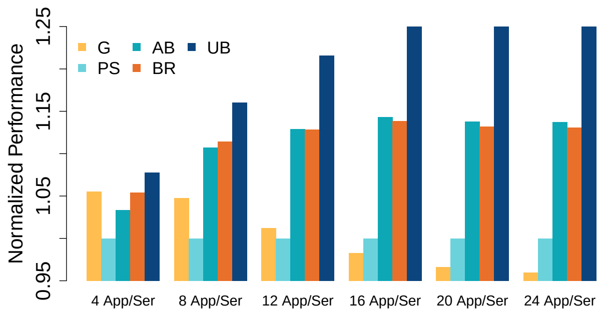

Figure 9: Average System Performance.

- **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 [40]. Since Amdahl utilities are concave, the interior point method finds globally optimal bids in polynomial time.

(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.

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.

B. System Performance

Figure 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 [6]. 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 (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 [12].

(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.

C. Entitlements and Performance

Figure 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 (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 §V and a similar one applies to (BR). Figure 10 presents the same finding empirically.

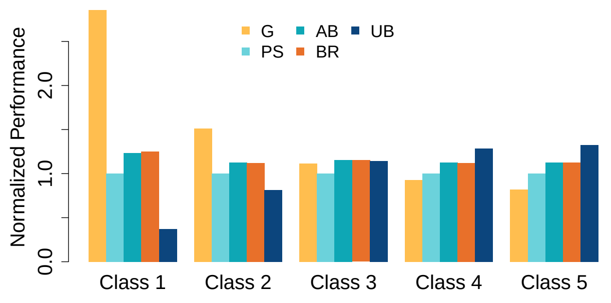

Figure 10: Per-class performance measured in terms of user utility.

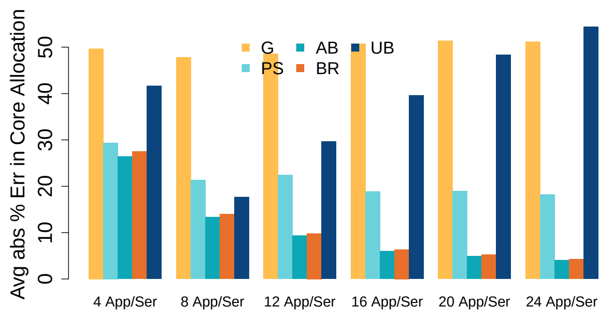

Figure 11: Mean Absolute Percentage Error (MAPE) of core allocations under different allocation policies and global core entitlements.

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.

D. Entitlements and Allocations

Figure 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

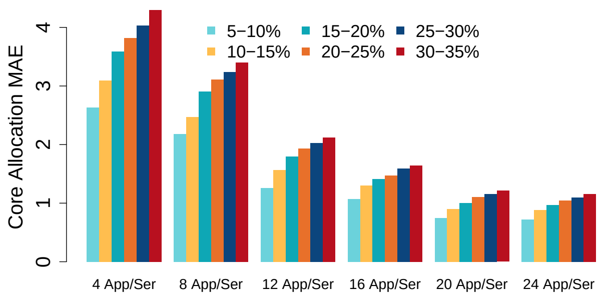

Figure 12: Mean Abs Err (MAE) in core allocation due to over-estimation of $\bar{F}$.

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 §II, 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. 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).

E. 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% [41]. Finally, we compare market allocations when using original and adjusted estimates.

Figure 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.

F. 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 larger share.

Figure 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.

VII. Related Work

Numerous studies cast hardware management as a market problem in which users bid for resources [12], [10], [42]. 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 [12]. 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

[5] 12.35ms = 10×(0.10ms + 0.85ms + 0.25ms) + 0.35ms.

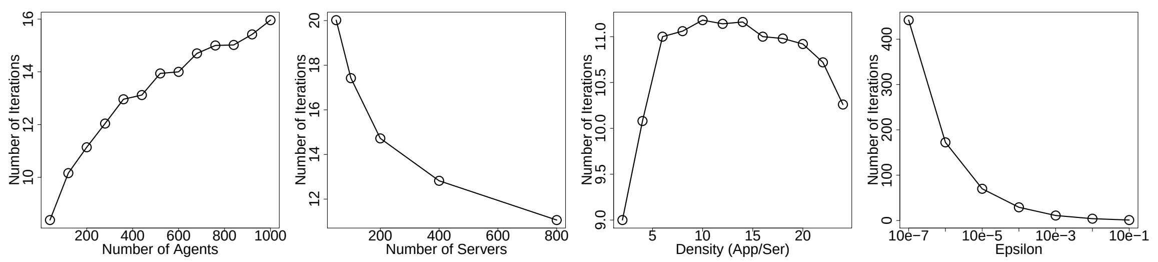

Figure 13: Convergence rate of (AB).

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 [17], [43], [19]. Zahedi et al. use the Cobb-Douglas utility function to allocate cache capacity and memory bandwidth [44]. 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 [45]. 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 [41].

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 [46]. 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 [47]. 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.

VIII. 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 [9].

Acknowledgments

This work is supported by the National Science Foundation under grants CCF-1149252 (CAREER), CCF-1337215 (XPS-CLCCA), SHF-1527610, and AF-1408784. This work is 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.

References

[1] J. Kay and P. Lauder, "A fair share scheduler," *Communications of the ACM*, vol. 31, no. 1, pp. 44–55, 1988.

[2] 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.

[3] 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.

[4] "Amazon EC2," https://aws.amazon.com/ec2/.

[5] "Microsoft azure," https://azure.microsoft.com.

[6] G. J. Henry, “The UNIX system: The fair share scheduler,” *AT&T Bell Laboratories Technical Journal*, vol. 63, no. 8, pp. 1845–1857, 1984.

[7] 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.

[8] 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.

[9] M. D. Hill and M. R. Marty, “Amdahl’s Law in the multicore era,” *Computer*, vol. 41, no. 7, 2008.

[10] M. Guevara, B. Lubin, and B. C. Lee, “Navigating heterogeneous processors with market mechanisms,” in *Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 95–106.

[11] ——, “Strategies for anticipating risk in heterogeneous system design,” in *Proceedings of the 20th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 154–164.

[12] 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.

[13] ——, “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.

[14] 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.

[15] J. A. Ang, R. A. Ballance, L. A. Fisk, J. R. Johnston, and K. T. Pedretti, “Red storm capability computing queuing policy,” 2005.

[16] Duke University, “The Duke Computer Cluster,” https://rc.duke.edu/the-duke-compute-cluster.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] “Web data commons: Hyperlink graphs,” http://webdatacommons.org/hyperlinkgraph/index.html.

[22] “US Census Data (1990) data set,” https://archive.ics.uci.edu/ml/datasets/US+Census+Data+(1990).

[23] “Movielens dataset,” http://grouplens.org/datasets/movielens/.

[24] 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.

[25] 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.

[26] A. Karp and H. Flatt, “Measuring parallel processor performance,” *Communications of the ACM*, 1990.

[27] C. Bienia, “Benchmarking modern multiprocessors,” Ph.D. dissertation, Princeton University, January 2011.

[28] “Docker,” http:/docs.docker.com.

[29] 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.

[30] 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.

[31] “Tycoon: Market-based resource allocation,” http://www.hpl.hp.com/research/tycoon/index.html, last visited: June 2017.

[32] B. N. Chun and D. E. Culler, “Market-based proportional resource sharing for clusters,” Berkeley, CA, USA, Tech. Rep., 2000.

[33] 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.

[34] 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.

[35] L. Zhang, “Proportional response dynamics in the Fisher market,” *Theoretical Computer Science*, vol. 412, no. 24, pp. 2691–2698, 2011.

[36] K. J. Arrow and G. Debreu, “Existence of an equilibrium for a competitive economy,” *Econometrica: Journal of the Econometric Society*, pp. 265–290, 1954.

[37] L. Zhang, “Proportional response dynamics in the Fisher market,” *Theoretical Computer Science*, vol. 412, no. 24, pp. 2691–2698, 2011.

[38] S. Eyerman and L. Eeckhout, “System-level performance metrics for multiprogram workloads,” *IEEE Micro*, vol. 28, no. 3, pp. 42–53, 2008.

[39] 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.

[40] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.