The Variable-Processor Cup Game

William Kuszmaul
CSAIL, MIT, Cambridge, MA, USA
kuszmaul@mit.edu

Alek Westover
CSAIL, MIT, Cambridge, MA, USA
alek.westover@gmail.com

Abstract

The problem of scheduling tasks on p processors so that no task ever gets too far behind is often described as a game with cups and water. In the p-processor cup game on n cups, there are two players, a filler and an emptier, that take turns adding and removing water from a set of n cups. In each turn, the filler adds p units of water to the cups, placing at most 1 unit of water in each cup, and then the emptier selects p cups to remove up to 1 unit of water from. The emptier's goal is to minimize the backlog, which is the height of the fullest cup.

The p-processor cup game has been studied in many different settings, dating back to the late 1960's. All of the past work shares one common assumption: that p is fixed. This paper initiates the study of what happens when the number of available processors p varies over time, resulting in what we call the *variable-processor cup game*.

Remarkably, the optimal bounds for the variable-processor cup game differ dramatically from its classical counterpart. Whereas the p-processor cup has optimal backlog $\Theta(\log n)$, the variable-processor game has optimal backlog $\Theta(n)$. Moreover, there is an efficient filling strategy that yields backlog $\Omega(n^{1-\varepsilon})$ in quasi-polynomial time against any deterministic emptying strategy.

We additionally show that straightforward uses of randomization cannot be used to help the emptier. In particular, for any positive constant Δ, and any Δ-greedy-like randomized emptying algorithm $\mathcal{A}$, there is a filling strategy that achieves backlog $\Omega(n^{1-\varepsilon})$ against $\mathcal{A}$ in quasi-polynomial time.

2012 ACM Subject Classification Theory of computation → Parallel algorithms

Keywords and phrases scheduling, cup games, online algorithms, lower bounds

Digital Object Identifier 10.4230/LIPIcs.ITCS.2021.16

Related Version `https://arxiv.org/abs/2012.00127`

Funding This research was sponsored in part by National Science Foundation Grants XPS-1533644 and CCF-1930579, and in part by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.
William Kuszmaul: Funded by a Hertz Fellowship and an NSF GRFP fellowship.

Acknowledgements Portions of this work were completed at the Second Hawaii Workshop on Parallel Algorithms and Data Structures. The authors would like to thank Nodari Sitchinava for organizing the workshop, and John Iacono for several helpful discussions relating to the variable-processor cup game.

1 Introduction

A fundamental challenge in processor scheduling is how to perform load balancing, that is, how to share processors among tasks in order to keep any one task from getting too far behind. Consider n tasks executing in time slices on $p < n$ processors. During each time slice,

a scheduler must select p (distinct) tasks that will be executed during the slice; up to one unit of work is completed on each executed task. During the same time slice, however, up to p units of *new work* may be allocated to the tasks, with different tasks receiving different amounts of work. The goal of a load-balancing scheduler is to bound the backlog of the system, which is defined to be the maximum amount of uncompleted work for any task.

As a convention, the load balancing problem is often described as a game involving water and cups. The ***p*-processor cup game** is a multi-round game with two players, an **emptier** and a **filler**, that takes place on n initially empty cups. At the beginning of each round, the filler adds up to p units of water to the cups, subject to the constraint that each cup receives at most 1 unit of water. The emptier then selects up to p distinct cups and removes up to 1 unit of water from each of them. The emptier's goal is to minimize the amount of water in the fullest cup, also known as the **backlog**. In terms of processor scheduling, the cups represent tasks, the water represents work assigned to each task, and the emptier represents a scheduling algorithm.

Starting with the seminal paper of Liu [31], work on the p processor cup game has spanned more than five decades [7, 21, 8, 30, 28, 34, 6, 24, 31, 32, 17, 10, 26, 1, 16, 29]. In addition to processor scheduling [7, 21, 8, 30, 28, 34, 6, 24, 31, 32, 1, 29, 17], applications include network-switch buffer management [22, 4, 36, 20], quality of service guarantees [7, 1, 29], and data structure deamortization [2, 17, 16, 3, 35, 23, 18, 25, 9].

The game has also been studied in many different forms. Researchers have studied the game with a fixed-filling-rate constraint [7, 21, 8, 30, 28, 34, 6, 24, 31, 32], with various forms of resource augmentation [10, 26, 29, 17], with both oblivious and adaptive adversaries [1, 7, 31, 10, 26, 11], with smoothed analysis [26, 10], with a semi-clairvoyant emptier [29], with competitive analysis [5, 19, 15], etc.

For the plain form of the p-processor cup game, the greedy emptying algorithm (i.e., always empty from the fullest cups) is known to be asymptotically optimal [1, 10, 26], achieving backlog $O(\log n)$. The optimal backlog for randomized emptying algorithms remains an open question [17, 10, 26] and is known to be between $\Omega(\log \log n)$ and $O(\log \log n + \log p)$ [26].

This paper: varying resources

Although cup games have been studied in many forms, all of the prior work on cup games shares one common assumption: the number p of processors is fixed.

In modern computing, however, computers are often shared among multiple applications, users, and even virtual OS's. The result is that the amount of resources (e.g., memory, processors, cache, etc.) available to a given application may fluctuate over time. The problem of handling cache fluctuations has received extensive attention in recent years (see work on cache-adaptive analysis [33, 12, 13, 14, 27]), but the problem of handling a varying number of processors remains largely unstudied.

This paper introduces the **variable-processor cup game**, in which the filler is allowed to *change* p (the total amount of water that the filler adds, and the emptier removes, from the cups per round) at the beginning of each round. Note that we do not allow the resources of the filler and emptier to vary separately. That is, as in the standard game, we take the value of p for the filler and emptier to be identical in each round. This restriction is crucial since, if the filler is allowed more resources than the emptier, then the filler could trivially achieve unbounded backlog.

A priori having variable resources offers neither player a clear advantage. When the number p of processors is fixed, the greedy emptying algorithm (i.e., always empty from the fullest cups), is known to achieve backlog $O(\log n)$ [1, 10, 26] regardless of the value of p.

This seems to suggest that, when p varies, the correct backlog should remain $O(\log n)$. In fact, when we began this project, we hoped for a straightforward reduction between the two versions of the game.

Results

We show that the variable-processor cup game is *not* equivalent to the standard p-processor game. By strategically controlling the number p of processors, the filler can achieve substantially larger backlog than would otherwise be possible.

We begin by constructing filling strategies against deterministic emptying algorithms. We show that for any positive constant ε, there is a filling strategy that achieves backlog $\Omega(n^{1-\varepsilon})$ within $2^{\text{polylog}(n)}$ rounds. Moreover, if we allow for $n!$ rounds, then there is a filling strategy that achieves backlog $\Omega(n)$. In contrast, for the p-processor cup game with any fixed p, the backlog never exceeds $O(\log n)$.

Our lower-bound construction is asymptotically optimal. By introducing a novel set of invariants, we deduce that the greedy emptying algorithm never lets backlog exceed $O(n)$.

A natural question is whether *randomized* emptying algorithms can do better. In particular, when the emptier is randomized, the filler is taken to be **oblivious**, meaning that the filler cannot see what the emptier does at each step. Thus the emptier can potentially use randomization to obfuscate their behavior from the filler, preventing the filler from being able to predict the heights of cups.

When studying randomized emptying strategies, we restrict ourselves to the class of **greedy-like emptying strategies**. This means that the emptier never chooses to empty from a cup c over another cup c' whose fill is more than $\omega(1)$ greater than the fill of c. All of the known randomized emptying strategies for the classic p-processor cup game are greedy-like [10, 26].

Remarkably, the power of randomization does not help the emptier very much in the variable-processor cup game. For any constant $\varepsilon > 0$, we give an oblivious filling strategy that achieves backlog $\Omega(n^{1-\varepsilon})$ in quasi-polynomial time (with probability $1 - 2^{-\text{polylog}\, n}$), no matter what (possibly randomized) greedy-like strategy the emptier follows.

Our results combine to tell a surprising story. They suggest that the problem of varying resources poses a serious theoretical challenge for the design and analysis of load-balancing scheduling algorithms. There are many possible avenues for future work. Can techniques from beyond worst-case analysis (e.g., smoothing, resource augmentation, etc.) be used to achieve better bounds on backlog? What about placing restrictions on the filler (e.g., bounds on how fast p can change), allowing the emptier to be able to be semi-clairvoyant, or making stochastic assumptions on the filler? We believe that all of these questions warrant further attention.

Paper outline

In Section 2 we establish the conventions and notations that we will use to discuss the variable-processor cup game. In Section 3 we analyze the greedy emptying algorithm, showing that it achieves backlog $O(n)$. We then turn our attention to designing (both oblivious and adaptive) filling strategies that achieve large backlog. Section 4 gives a technical overview of the filling strategies and their analyses. Section 5 then gives a full treatment of the filling strategies in the case where the filler is adaptive; the full treatment of the case where the filler is oblivious is deferred to the extended version of this paper.

2 Preliminaries

The cup game consists of a sequence of rounds. Let S_t denote the state of of the cups at the start of round t. At the beginning of the round, the filler chooses the number of processors p_t for the round. Next, the filler distributes p_t units of water among the cups (with at most 1 unit of water to any particular cup). Now the game is at the **intermediate state** for round t, which we call state I_t. Finally the emptier chooses p_t cups to empty at most 1 unit of water from, which marks the conclusion of round t. The state is then S_{t+1}.

If the emptier empties from a cup c on round t such that the fill of c is less than 1 in state I_t, then c now has 0 fill (not negative fill); we say that the emptier **zeroes out** c on round t. Note that on any round where the emptier zeroes out a cup the emptier has removed less total fill from the cups than the filler has added to the cups; hence the average fill of the cups has increased.

We denote the fill of a cup c at state S by $\text{fill}_S(c)$. Let the **mass** of a set of cups X at state S be $m_S(X) = \sum_{c \in X} \text{fill}_S(c)$. Denote the average fill of a set of cups X at state S by $\mu_S(X)$.[1] Note that $\mu_S(X)|X| = m_S(X)$. Let the **backlog** at state S be $\max_c \text{fill}_S(c)$, let the **anti-backlog** at state S be $\min_c \text{fill}_S(c)$. Let the **rank** of a cup at a given state be its position in the list of the cups sorted by fill at the given state, breaking ties arbitrarily but consistently. For example, the fullest cup at a state has rank 1, and the least full cup has rank n. Let $[n] = \{1, 2, \ldots, n\}$, let $i + [n] = \{i + 1, i + 2, \ldots, i + n\}$. For a state S, let $S(r)$ denote the rank r cup at state S, and let $S(\{r_1, r_2, \ldots, r_m\})$ denote the set of cups of ranks $r_1, r_2, \ldots r_m$.

As a tool in the analysis we define a new variant of the cup game: the **negative-fill** cup game. In the negative-fill cup game, when the emptier empties from a cup, the cup's fill always decreases by exactly 1, i.e. there is no zeroing out. We refer to the standard version of the cup game where cups can zero out as the **standard-fill** cup game.

The notion of negative fill will be useful in our lower-bound constructions, in which we want to construct a strategy for the filler that achieves large backlog. By analyzing a filling strategy on the negative-fill game, we can then reason about what happens if we apply the same filling strategy recursively to a set of cups S whose average fill μ is larger than 0; in the recursive application, the average fill μ acts as the "new 0", and fills less than μ act as negative fills.

Note that it is strictly easier for the filler to achieve high backlog in the standard-fill cup game than in the negative-fill cup game; hence a lower bound on backlog in the negative-fill cup game also serves as a lower bound on backlog in the standard-fill cup game. On the other hand, during the upper bound proof we use the standard-fill cup game: this makes it harder for the emptier to guarantee its upper bound.

Other Conventions

When discussing the state of the cups **at a round t**, we will take it as given that we are referring to the starting state S_t of the round. Also, when discussing sets, we will use XY as a shorthand for $X \cup Y$. Finally, when discussing the average fill $\mu_{S_t}(X)$ of a set of cups, we will sometimes ommit the subscript S_t when the round number is clear.

[1] For both mass and average fill, in cases where S is clear from context, we may omit the subscript, writing $m(X)$ or $\mu(X)$.

3 Upper Bound

In this section we analyze the **greedy emptier**, which always empties from the p fullest cups. We prove in Corollary 2 that the greedy emptier prevents backlog from exceeding $O(n)$. In order to analyze the greedy emptier, we establish a system of invariants that hold at every step of the game.

The main result of the section is the following theorem.

▶ **Theorem 1.** *In the variable-processor cup game on n cups, the greedy emptier maintains, at every step t, the invariants*

$$\mu_{S_t}(S_t([k])) \leq 2n - k \tag{1}$$

for all $k \in [n]$.

By applying Theorem 1 to the case of $k = 1$, we arrive at a bound on backlog:

▶ **Corollary 2.** *In the variable-processor cup game on n cups, the greedy emptying strategy never lets backlog exceed $O(n)$.*

Proof of Theorem 1. We prove the invariants by induction on t. The invariants hold trivially for $t = 1$ (the base case for the inductive proof): the cups start empty so $\mu_{S_1}(S_1([k])) = 0 \leq 2n - k$ for all $k \in [n]$.

Fix a round $t \geq 1$, and any $k \in [n]$. We assume the invariant for all values of $k' \in [n]$ for state S_t (we will only explicitly use two of the invariants for each k, but the invariants that we need depend on the choice of p_t by the filler) and show that the invariant on the k fullest cups holds on round $t + 1$, i.e. that

$$\mu_{S_{t+1}}(S_{t+1}([k])) \leq 2n - k.$$

Note that because the emptier is greedy it always empties from the cups $I_t([p_t])$. Let A, with $a = |A|$, be $A = I_t([\min(k, p_t)]) \cap S_{t+1}([k])$; A consists of the cups that are among the k fullest cups in I_t, are emptied from, and are among the k fullest cups in S_{t+1}. Let B, with $b = |B|$, be $I_t([\min(k, p_t)]) \setminus A$; B consists of the cups that are among the k fullest cups in state I_t, are emptied from, and are not among the k fullest cups in S_{t+1}. Let $C = I_t(a+b+[k-a])$, with $c = k - a = |C|$; C consists of the cups with ranks $a + b + 1, \ldots, k + b$ in state I_t. The set C is defined so that $S_{t+1}([k]) = AC$, since once the cups in B are emptied from, the cups in B are not among the k fullest cups, so cups in C take their places among the k fullest cups.

Note that $k - a \geq 0$ as $a + b \leq k$, and also $|ABC| = k + b \leq n$, because by definition the b cups in B must not be among the k fullest cups in state S_{t+1} so there are at least $k + b$ cups. Note that $a + b = \min(k, p_t)$. We also have that $A = I_t([a])$ and $B = I_t(a + [b])$, as every cup in A must have higher fill than all cups in B in order to remain above the cups in B after 1 unit of water is removed from all cups in AB.

We now establish the following claim, which we call the **interchangeability of cups**:

▷ Claim 3. There exists a cup state S'_t such that: (a) S'_t satisfies the invariants (1), (b) $S'_t(r) = I_t(r)$ for all ranks $r \in [n]$, and (c) the filler can legally place water into cups in order to transform S'_t into I_t.

Proof. Fix $r \in [n]$. We will show that S_t can be transformed into a state S^r_t by relabelling only cups with ranks in $[r]$ such that (a) S^r_t satisfies the invariants (1), (b) $S^r_t([r]) = I_t([r])$ and (c) the filler can legally place water into cups in order to transform S^r_t into I_t.

Say there are cups x, y with $x \in S_t([r]) \setminus I_t([r]), y \in I_t([r]) \setminus S_t([r])$. Let the fills of cups x, y at state S_t be f_x, f_y; note that

$$f_x > f_y. \tag{2}$$

Let the amount of fill that the filler adds to these cups be $\Delta_x, \Delta_y \in [0, 1]$; note that

$$f_x + \Delta_x < f_y + \Delta_y. \tag{3}$$

Define a new state S'_t where cup x has fill f_y and cup y has fill f_x. Note that the filler can transform state S'_t into state I_t by placing water into cups as before, except changing the amount of water placed into cups x and y to be $f_x - f_y + \Delta_x$ and $f_y - f_x + \Delta_y$, respectively.

In order to verify that the transformation from S'_t to I_t is a valid step for the filler, one must check three conditions. First, the amount of water placed by the filler is unchanged: this is because $(f_x - f_y + \Delta_x) + (f_y - f_x + \Delta_y) = \Delta_x + \Delta_y$. Second, the fills placed in cups x and y are at most 1: this is because $f_x - f_y + \Delta_x < \Delta_y \leq 1$ (by (3)) and $f_y - f_x + \Delta_x < \Delta_x \leq 1$ (by (2)). Third, the fills placed in cups x and y are non-negative: this is because $f_x - f_y + \Delta_x > \Delta_x \geq 0$ (by (2)) and $f_y - f_x + \Delta_y > \Delta_x \geq 0$ (by (3)).

We can repeatedly apply this process to swap each cup in $I_t([r]) \setminus S_t([r])$ into being in $S'_t([r])$. At the end of this process we will have some state S^r_t for which $S^r_t([r]) = I_t([r])$. Note that S^r_t is simply a relabeling of S_t, hence it must satisfy the same invariants (1) satisfied by S_t. Further, S^r_t can be transformed into I_t by a valid filling step.

Now we repeatedly apply this process, in descending order of ranks. In particular, we have the following process: create a sequence of states by starting with S^{n-1}_t, and to get to state S^r_t from state S^{r+1}_t apply the process described above. Note that S^{n-1}_t satisfies $S^{n-1}_t([n-1]) = I_t([n-1])$ and thus also $S^{n-1}_t(n) = I_t(n)$. If S^{r+1}_t satisfies $S^{r+1}_t(r') = I_t(r')$ for all $r' > r+1$ then S^r_t satisfies $S^r_t(r') = I_t(r')$ for all $r > r$, because the transition from S^{r+1}_t to S^r_t has not changed the labels of any cups with ranks in $(r+1, n]$, but the transition does enforce $S^r_t([r]) = I_t([r])$, and consequently $S^r_t(r+1) = I_t(r+1)$. We continue with the sequential process until arriving at state S^1_t in which we have $S^1_t(r) = I_t(r)$ for all r. Throughout the process each S^r_t has satisfied the invariants (1), so S^1_t satisfies the invariants (1). Further, throughout the process from each S^r_t it is possible to legally place water into cups in order to transform S^r_t into I_t.

Hence S^1_t satisfies all the properties desired, and the proof of Claim 3 is complete. ◁

Claim 3 tells us that we may assume without loss of generality that $S_t(r) = I_t(r)$ for each rank $r \in [n]$. We will make this assumption for the rest of the proof.

In order to complete the proof of the theorem, we break it into three cases.

▷ Claim 4. If some cup in A zeroes out in round t, then the invariant $\mu_{S_{t+1}}(S_{t+1}([k])) \leq 2n-k$ holds.

Proof. Say a cup in A zeroes out in step t. Of course

$$m_{S_{t+1}}(I_t([a-1])) \leq (a-1)(2n-(a-1))$$

because the $a-1$ fullest cups must have satisfied the invariant (with $k = a-1$) on round t. Moreover, because $\text{fill}_{S_{t+1}}(I_{t+1}(a)) = 0$

$$m_{S_{t+1}}(I_t([a])) = m_{S_{t+1}}(I_t([a-1])).$$

Combining the above equations, we get that

$$m_{S_{t+1}}(A) \leq (a-1)(2n-(a-1)).$$

Furthermore, the fill of all cups in C must be at most 1 at state I_t to be less than the fill of the cup in A that zeroed out. Thus,

$$\begin{aligned} m_{S_{t+1}}(S_{t+1}([k])) &= m_{S_{t+1}}(AC) \\ &\leq (a-1)(2n-(a-1)) + k - a \\ &= a(2n-a) + a - 2n + a - 1 + k - a \\ &= a(2n-a) + (k-n) + (a-n) - 1 \\ &< a(2n-a) \end{aligned}$$

as desired. As k increases from 1 to n, $k(2n-k)$ strictly increases (it is a quadratic in k that achieves its maximum value at $k=n$). Thus $a(2n-a) \leq k(2n-k)$ because $a \leq k$. Therefore,

$$m_{S_{t+1}}(S_{t+1}([k])) \leq k(2n-k). \qquad \triangleleft$$

▷ Claim 5. If no cups in A zero out in round t and $b=0$, then the invariant $\mu_{S_{t+1}}(S_{t+1}([k])) \leq 2n-k$ holds.

Proof. If $b=0$, then $S_{t+1}([k]) = S_t([k])$. During round t the emptier removes a units of fill from the cups in $S_t([k])$, specifically the cups in A. The filler cannot have added more than k fill to these cups, because it can add at most 1 fill to any given cup. Also, the filler cannot have added more than p_t fill to the cups because this is the total amount of fill that the filler is allowed to add. Hence the filler adds at most $\min(p_t, k) = a + b = a + 0 = a$ fill to these cups. Thus the invariant holds:

$$m_{S_{t+1}}(S_{t+1}([k])) \leq m_{S_t}(S_t([k])) + a - a \leq k(2n-k). \qquad \triangleleft$$

The remaining case, in which no cups in A zero out and $b > 0$ is the most technically interesting.

▷ Claim 6. If no cups in A zero out on round t and $b > 0$, then the invariant $\mu_{S_{t+1}}(S_{t+1}([k])) \leq 2n-k$ holds.

Proof. Because $b > 0$ and $a + b \leq k$ we have that $a < k$, and $c = k - a > 0$. Recall that $S_{t+1}([k]) = AC$, so the mass of the k fullest cups at S_{t+1} is the mass of AC at S_t plus any water added to cups in AC by the filler, minus any water removed from cups in AC by the emptier. The emptier removes exactly a units of water from AC. The filler adds no more than p_t units of water to AC (because the filler adds at most p_t total units of water per round) and the filler also adds no more than $k = |AC|$ units of water to AC (because the filler adds at most 1 unit of water to each of the k cups in AC). Thus, the filler adds no more than $a + b = \min(p_t, k)$ units of water to AC. Combining these observations we have:

$$m_{S_{t+1}}(S_{t+1}([k])) \leq m_{S_t}(AC) + b. \tag{4}$$

The key insight necessary to bound this is to notice that larger values for $m_{S_t}(A)$ correspond to smaller values for $m_{S_t}(C)$ because of the invariants; the higher fill in A **pushes down** the fill that C can have. By capturing the pushing-down relationship combinatorially we will achieve the desired inequality.

We can upper bound $m_{S_t}(C)$ by

$$\begin{aligned} m_{S_t}(C) &\leq \frac{c}{b+c} m_{S_t}(BC) \\ &= \frac{c}{b+c}(m_{S_t}(ABC) - m_{S_t}(A)) \end{aligned}$$

because $\mu_{S_t}(C) \leq \mu_{S_t}(B)$ without loss of generality by the interchangeability of cups. Thus we have

$$m_{S_t}(AC) \leq m_{S_t}(A) + \frac{c}{b+c} m_{S_t}(BC) \tag{5}$$

$$= \frac{c}{b+c} m_{S_t}(ABC) + \frac{b}{b+c} m_{S_t}(A). \tag{6}$$

Note that the expression in (6) is monotonically increasing in both $\mu_{S_t}(ABC)$ and $\mu_{S_t}(A)$. Thus, by numerically replacing both average fills with their extremal values, $2n - |ABC|$ and $2n - |A|$. At this point the claim can be verified by straightforward (but quite messy) algebra (and by combining (4) with (6)). We instead give a more intuitive argument, in which we examine the right side of (5) combinatorially.

Consider a new configuration of fills F achieved by starting with state S_t, and moving water from BC into A until $\mu_F(A) = 2n - |A|$. [2] This transformation increases (strictly increases if and only if we move a non-zero amount of water) the right side of (5). In particular, if mass $\Delta \geq 0$ fill is moved from BC to A, then the right side of (5) increases by $\frac{b}{b+c}\Delta \geq 0$. Note that the fact that moving water from BC into A increases the right side of (5) formally captures the way the system of invariants being proven forces a tradeoff between the fill in A and the fill in BC – that is, higher fill in A pushes down the fill that BC (and consequently C) can have.

Since $\mu_F(A)$ is above $\mu_F(ABC)$, the greater than average fill of A must be counterbalanced by the lower than average fill of BC. In particular we must have

$$(\mu_F(A) - \mu_F(ABC))|A| = (\mu_F(ABC) - \mu_F(BC))|BC|.$$

Note that

$$\begin{aligned}
&\mu_F(A) - \mu_F(ABC) \\
&= (2n - |A|) - \mu_F(ABC) \\
&\geq (2n - |A|) - (2n - |ABC|) \\
&= |BC|.
\end{aligned}$$

Hence we must have

$$\mu_F(ABC) - \mu_F(BC) \geq |A|.$$

Thus

$$\mu_F(BC) \leq \mu_F(ABC) - |A| \leq 2n - |ABC| - |A|. \tag{7}$$

Combing (5) with the fact that the transformation from S_t to F only increases the right side of (5), along with (7), we have the following bound:

$$\begin{aligned}
m_{S_t}(AC) &\leq m_F(A) + c\mu_F(BC) \\
&\leq a(2n - a) + c(2n - |ABC| - a) \\
&\leq (a + c)(2n - a) - c(a + c + b) \\
&\leq (a + c)(2n - a - c) - cb.
\end{aligned} \tag{8}$$

[2] Note that whether or not F satisfies the invariants is irrelevant.

By (4) and (8), we have that

$$\begin{aligned} m_{S_{t+1}}(S_{t+1}([k])) &\leq m_{S_t}(AC) + b \\ &\leq (a+c)(2n-a-c) - cb + b \\ &= k(2n-k) - cb + b \\ &\leq k(2n-k), \end{aligned}$$

where the final inequality uses the fact that $c \geq 1$. This completes the proof of the claim. ◁

We have shown the invariant holds for arbitrary k, so given that the invariants all hold at state S_t they also must all hold at state S_{t+1}. Thus, by induction we have the invariant for all rounds $t \in \mathbb{N}$. ◀

4 Technical overview of lower bounds

The rest of the paper is devoted to the construction of filler strategies that match (or nearly match) the upper bound in Section 3.

This section gives a technical overview of the main technical ideas used in the filler constructions. Full versions of the constructions are given in Section 5 (for adaptive fillers) and in the extended version of this paper (for oblivious fillers).

4.1 Adaptive Lower Bound

In Section 5 we provide an adaptive filling strategy that achieves backlog $\Omega(n^{1-\varepsilon})$; in this subsection we sketch the proof of the result.

First we note that there is a trivial algorithm, that we call **trivalg**, for achieving backlog at least $1/2$ on at least 2 cups in time $O(1)$.

The essential ingredient in our lower-bound construction is what we call the **Amplification Lemma**. The lemma takes as input a filling strategy that achieves some backlog curve f (i.e., on n cups, the strategy achieves backlog $f(n)$), and outputs a new filling strategy that achieves a new amplified backlog curve f'.

▶ **Lemma 7** (Lemma 12). *Let* $\text{alg}(f)$ *be a filling strategy that achieves backlog* $f(n)$ *on* n *cups (in the negative-fill cup game). There exists a filling strategy* $\text{alg}(f')$*, the* ***amplification*** *of* $\text{alg}(f)$*, that achieves backlog at least*

$$f'(n) \geq (1-\delta)f(\lfloor(1-\delta)n\rfloor) + f(\lceil\delta n\rceil).$$

Proof Sketch. The filler designates an **anchor set** A of size $\lceil\delta n\rceil$ and a **non-anchor set** B of size $\lfloor(1-\delta)n\rfloor$.

The filler's strategy begins with M phases, for some parameter M to be determined later. In each phase, the filler applies $\text{alg}(f)$ to the non-anchor set B, while simultaneously placing 1 unit of water into each cup of A on each step. If there is ever a step during the phase in which the emptier does not remove water from every cup in A, then the phase is said to be **emptier neglected**. On the other hand, if a phase is not emptier neglected, then at the end of the phase, the filler swaps the cup in B with largest fill with the cup in A whose fill is smallest.

After the M phases are complete, the filler then recursively applies $\text{alg}(f)$ to the cups A. This completes the filling strategy $\text{alg}(f')$.

The key to analyzing $\text{alg}(f')$ is to show that, at the end of the M-th phase, the average fill of the cups A satisfies $\mu(A) \geq (1-\delta)f(|B|)$. This, in turn, means that the recursive application of $\text{alg}(f)$ to A will achieve backlog $(1-\delta)f(|B|) + f(|A|)$, as desired.

Now let us reason about $\mu(A)$. If a phase is emptier neglected, then the total amount of water placed into A during the phase is at least 1 greater than the total amount of water removed. Hence $\mu(A)$ increases by at least $1/|A|$. On the other hand, if a phase is not emptier neglected, then $\text{alg}(f)$ will successfully achieve backlog $\mu(B) + f(|B|)$ on the cups B during the phase. At the end of the phase, the filler will then swap a cup from B with large fill with a cup from A. Thus, in each phase, we either have that $\mu(A)$ increases by $1/|A|$, or that a cup with large fill gets swapped into A. After sufficiently many phases, one can show that $\mu(A)$ is guaranteed to become at least $(1-\delta)f(|B|) + f(|A|)$. ◀

We use the Amplification Lemma to give two lower bounds on backlog: one with reasonable running time, the other with slightly better backlog.

▶ **Theorem 8** (Theorem 13). *There is an adaptive filling strategy for achieving backlog $\Omega(n^{1-\varepsilon})$ for constant $\varepsilon \in (0, 1/2)$ in running time $2^{O(\log^2 n)}$.*

Proof Sketch. We construct a sequence of filling strategies with $\text{alg}(f_{i+1})$ the amplification of $\text{alg}(f_i)$ using $\delta = \Theta(1)$ determined as a function of ε, and $\text{alg}(f_0) = \text{trivalg}$. Choosing δ appropriately as a function of ε, and letting c be some (small) positive constant, we show by induction on i that, for all $k \leq 2^{ci}$, $\text{alg}(f_i)$ achieves backlog $\Omega(k^{1-\varepsilon})$ on k cups in running time $2^{O(\log^2 k)}$. Taking $i = \Theta(\log n)$ completes the proof. ◀

▶ **Theorem 9** (Theorem 15). *There is an adaptive filling strategy for achieving backlog $\Omega(n)$ in running time $O(n!)$.*

Proof Sketch. We construct a sequence of filling strategies with $\text{alg}(f_{i+1})$ the amplification of $\text{alg}(f_i)$ using $\delta = 1/(i+1)$, and $\text{alg}(f_0)$ a filling strategy for achieving backlog 1 on $O(1)$ cups in $O(1)$ time (this is a slight modification of trivalg). We show by induction that $\text{alg}(f_{\Theta(n)})$ achieves backlog $\Omega(n)$ in running time $O(n!)$. ◀

4.2 Oblivious Lower Bound

We now consider what happens if the filler is an **oblivious** adversary, meaning that the filler cannot see what the emptier does at each step. The emptier, in turn, is permitted to use randomization in order to make its behavior unpredictable to the filler. We focus on randomized emptying algorithms that satisfy the so-called **Δ-greedy-like** property: the emptier never empties from a cup c over another cup c' whose fill is more than Δ greater than the fill of c.

The next theorem gives an oblivious filling strategy that achieves backlog $\Omega(n^{1-\varepsilon})$ against any Δ-greedy-like emptier for any $\Delta \in \Omega(1)$ (or, more precisely, any $\Delta \leq \frac{1}{128}\log\log\log n$).

▶ **Theorem 10.** *There is an oblivious filling strategy for the variable-processor cup game on N cups that achieves backlog at least $\Omega(N^{1-\varepsilon})$ for any constant $\varepsilon > 0$ in running time $2^{polylog(N)}$ with probability at least $1 - 2^{-polylog(N)}$ against a Δ-greedy-like emptier with $\Delta \leq \frac{1}{128}\log\log\log N$.*

Note that Theorem 10 uses N for the number of cups rather than using n. When describing the recursive strategy that the filler uses, we will use N to denote the true total number of cups, and n to denote the number of cups within the recursive subproblem currently being discussed.

The filling strategy used in Theorem 10 is closely related the adaptive filling strategy described in Section 4.1. The fact that the filling strategy must now be *oblivious*, however, introduces several new technical obstacles.

Problem 1: Distinguishing between neglected and non-neglected phases

Recall that the Amplification Lemma in Section 4.1 proceeds in phases, where the filler behaves differently at the end of each phase depending on whether or not the emptier ever neglected the anchor set A during that phase. If the filler is oblivious, however, then it cannot detect which phases are neglected.

To solve this problem, the first thing to notice is that the *total* number of times that the emptier can neglect the anchor set (within a given recursive subproblem of the Amplification Lemma) is, without loss of generality, at most N^2. Indeed, if the emptier neglects the anchor set more than N^2 times, then the total amount of water in cups A will be at least N^2. Since the amount of water in the system as a whole is non-decreasing, there will subsequently always be at least one cup in the system with fill N or larger, and thus the filler's strategy trivially achieves backlog N.

Assume that there are at most N^2 phases that the emptier neglects. The filler does not know which phases these are, and the filler does not wish to ever move a cup from the non-anchor set to the anchor set during a phase that the emptier neglected (since, during such a phase, there is no guarantee on the amount of water in the cup from B). To solve this problem, we increase the total number of phases in the Amplification Lemma to be some very large number $M = 2^{\text{polylog}\, N}$, and we have the filler select $|A|$ random phases at the end of which to move a cup from the non-anchor set to the anchor set. With high probability, none of the $|A|$ phases that the filler selects are neglected by the emptier.

Problem 2: Handling the probability of failure

Because the filler is now oblivious (and the emptier is randomized) the guarantee offered by the filling strategy is necessarily probabilistic. This makes the Amplification Lemma somewhat more difficult, since each application of $\text{alg}(f)$ now has some probability of failure.

We ensure that the applications of $\text{alg}(f)$ succeed with such high probability that we can ignore the possibility of any of them failing on phases when we need them to succeed. This necessitates making sure that the base-case construction $\text{alg}(f_0)$ succeeds with very high probability.

Fortunately, we can take a base-case construction alg_0 that succeeds with only constant (or even sub-constant) probability, and perform an Amplification-Lemma-like construction in order to obtain a new filling strategy alg_1 that achieves slightly *smaller* backlog, but that has a very high probability of succeeding.

To construct alg_1, we begin by performing the Amplification-Lemma construction on alg_0, but without recursing after the final phase. Even though many of the applications of alg_0 may fail, with high probability at least one application succeeds. This results in some cup c_* in A having high fill. Unfortunately, the filler does not know which cup has high fill, so it cannot simply take c_*. What the filler can do, however, is select some cup c, decrease the number of processors to 1, and then spend a large number of steps simply placing 1 unit of water into cup c in each step. By the Δ-greediness of the emptier, the emptier is guaranteed to focus on emptying from cup c_* (rather than cup c) until c attains large fill. This allows for the filler to obtain a cup c that the filler *knows* contains a large amount of water (with high probability). We use this approach to construct a base-case filling strategy $\text{alg}(f_0')$ that succeeds with high probability.

Problem 3: Non-flat starting states

The next problem that we encounter is that, at the beginning of any given phase, the cups in the non-anchor set may not start off with equal heights. Instead, some cups may contain very large amounts of water while others contain very small (and even negative) amounts of water[3]. This is not a problem for an adaptive filler, since the filler knows which cups contain small/large amounts of water, but it is a problem for an oblivious filler.

To avoid the scenario in which the cups in B are highly unequal, we begin each phase by first performing a **flattening construction** on the cups B, which causes the cups in B to all have roughly equal fills (up to $\pm O(\Delta)$). The flattening construction uses the fact that the emptier is Δ-greedy-like to ensure that cups which are overly full get flattened out by the emptier.

Putting the pieces together

By combining the ideas above, as well as handling other issues that arise (e.g., one must be careful to ensure that the average fills of A and B do not drift apart in unpredictable ways), one can prove Theorem 10.

5 Adaptive Filler Lower Bound

In this section we give a $2^{\text{polylog}\, n}$-time filling strategy that achieves backlog $n^{1-\varepsilon}$ for any positive constant ε. We also give a $O(n!)$-time filling strategy that achieves backlog $\Omega(n)$. These results formalize the ideas described in Section 4.1.

We begin with a trivial filling strategy that we call **trivalg** that gives backlog at least $1/2$ when applied to at least 2 cups.

▶ **Proposition 11.** *Consider an instance of the negative-fill 1-processor cup game on n cups, and let the cups start in any state with average fill is 0. If $n \geq 2$, there is an $O(1)$-step adaptive filling strategy* trivalg *that achieves backlog at least $1/2$. If $n = 1$,* trivalg *achieves backlog 0 in running time 0.*

Proof. If $n = 1$, trivalg does nothing and achieves backlog 0; for the rest of the proof we consider the case $n \geq 2$.

Let a and b be the fullest and second fullest cups in the in the starting configuration, and let their initial fills be $\text{fill}(a) = \alpha, \text{fill}(b) = \beta$. If $\alpha \geq 1/2$ the filler need not do anything, the desired backlog is already achieved. Otherwise, if $\alpha \in [0, 1/2]$, the filler places $1/2 - \alpha$ fill into a and $1/2 + \alpha$ fill into b (which is possible as both fills are in $[0, 1]$, and they sum to 1). Since $\alpha + \beta \geq 0$ we have $\beta \geq -\alpha$. Clearly a and b now both have fill at least $1/2$. The emptier cannot empty from both a and b as $p = 1$, so even after the emptier empties from a cup we still have backlog $1/2$, as desired. ◀

Next we prove the **Amplification Lemma**, which takes as input a filling strategy $\text{alg}(f)$ and outputs a new filling strategy $\text{alg}(f')$ that we call the **amplification** of $\text{alg}(f)$. $\text{alg}(f')$ is able to achieve higher fill than $\text{alg}(f)$; in particular, we will show that by starting with a filling strategy $\text{alg}(f_0)$ for achieving constant backlog and then forming a sufficiently long sequence of filling strategies $\text{alg}(f_0), \text{alg}(f_1), \ldots, \text{alg}(f_{i_*})$ with $\text{alg}(f_{i+1})$ the amplification of $\text{alg}(f_i)$, we get a filling strategy for achieving $\text{poly}(n)$ backlog.

[3] Recall that, in order for our lower-bound construction to be able to call itself recursively, we must analyze the construction in the negative-fill version of the variable-processor cup game.

▶ **Lemma 12** (Adaptive Amplification Lemma). *Let $\delta \in (0, 1/2]$ be a parameter. Let $\text{alg}(f)$ be an adaptive filling strategy that achieves backlog $f(n) < n$ in the negative-fill variable-processor cup game on n cups in running time $T(n)$ starting from any initial cup state where the average fill is 0.*

Then there exists an adaptive filling strategy $\text{alg}(f')$ that achieves backlog $f'(n)$ satisfying

$$f'(n) \geq (1-\delta)f(\lfloor(1-\delta)n\rfloor) + f(\lceil\delta n\rceil)$$

and $f'(n) \geq f(n)$ in the negative-fill variable-processor cup game on n cups in running time

$$T'(n) \leq n\lceil\delta n\rceil \cdot T(\lfloor(1-\delta)n\rfloor) + T(\lceil\delta n\rceil)$$

starting from any initial cup state where the average fill is 0.

Proof. Let $n_A = \lceil\delta n\rceil, n_B = n - n_A = \lfloor(1-\delta)n\rfloor$.

The filler defaults to using $\text{alg}(f)$ if

$$f(n) \geq (1-\delta)f(n_B) + f(n_A).$$

In this case using $\text{alg}(f)$ achieves the desired backlog in the desired running time. In the rest of the proof, we describe our strategy for the case where we cannot simply use $\text{alg}(f)$ to achieve the desired backlog.

Let A, the **anchor set**, be initialized to consist of the n_A fullest cups, and let B the **non-anchor set** be initialized to consist of the rest of the cups (so $|B| = n_B$). Let $h = (1-\delta)f(n_B)$.

The filler's strategy can be summarized as follows:
Step 1: Make $\mu(A) \geq h$ by using $\text{alg}(f)$ repeatedly on B to achieve cups with fill at least $\mu(B) + f(n_B)$ in B and then swapping these into A. While doing this the filler always places 1 unit of fill in each anchor cup.
Step 2: Use $\text{alg}(f)$ once on A to obtain some cup with fill $\mu(A) + f(n_A)$.
Note that in order to use $\text{alg}(f)$ on subsets of the cups the filler will need to vary p.

We now describe how to achieve Step 1, which is complicated by the fact that the emptier may attempt to prevent the filler from achieving high fill in a cup in B.

The filling strategy always places 1 unit of water in each anchor cup. This ensures that no cups in the anchor set ever have their fill decrease. If the emptier wishes to keep the average fill of the anchor cups from increasing, then emptier must empty from every anchor cup on each step. If the emptier fails to do this on a given round, then we say that the emptier has **neglected** the anchor cups.

We say that the filler **applies** $\text{alg}(f)$ to B if it follows the filling strategy $\text{alg}(f)$ on B while placing 1 unit of water in each anchor cup. An application of $\text{alg}(f)$ to B is said to be **successful** if A is never neglected during the application of $\text{alg}(f)$ to B. The filler uses a series of phases that we call **swapping-processes** to achieve the desired average fill in A. In a swapping-process, the filler repeatedly applies $\text{alg}(f)$ to B until a successful application occurs, and then takes the cup generated by $\text{alg}(f)$ within B on this successful application with fill at least $\mu(B) + f(|B|)$ and swaps it with the least full cup in A so long as the swap increases $\mu(A)$. If the average fill in A ever reaches h, then the algorithm immediately halts (even if it is in the middle of a swapping-process) and is complete.

We give pseudocode for the filling strategy in Algorithm 1.

Algorithm 1 Adaptive Amplification (Step 1).

Input: alg$(f), \delta$, set of n cups
Output: Guarantees that $\mu(A) \geq h$

$A \leftarrow n_A$ fullest cups, $B \leftarrow$ rest of the cups
Always place 1 fill in each cup in A
while $\mu(A) < h$ **do** ▷ Swapping-Processes
  Immediately **exit** this loop if ever $\mu(A) \geq h$
  successful $\leftarrow$ false
  while not successful **do**
    Apply alg(f) to B, alg(f) gives cup c
    if fill$(c) \geq h$ **then**
      successful $\leftarrow$ true
  Swap c with least full cup in A

Note that

$$\mu(A) \cdot |A| + \mu(B) \cdot |B| = \mu(AB) \geq 0,$$

as $\mu(AB)$ starts as 0, but could become positive if the emptier skips emptyings. Thus we have

$$\mu(A) \geq -\mu(B) \cdot \frac{\lfloor (1-\delta)n \rfloor}{\lceil \delta n \rceil} \geq -\frac{1-\delta}{\delta}\mu(B).$$

Thus, if at any point B has average fill lower than $-h \cdot \delta/(1-\delta)$, then A has average fill at least h, so the algorithm is finished. Thus we can assume in our analysis that

$$\mu(B) \geq -h \cdot \delta/(1-\delta). \tag{9}$$

We will now show that the filler applies alg(f) to B at most hn_A total times. Each time the emptier neglects the anchor set, the mass of the anchor set increases by 1. If the emptier neglects the anchor set hn_A times, then the average fill in the anchor set increases by h. Since $\mu(A)$ started as at least 0, and since $\mu(A)$ never decreases (note in particular that cups are only swapped into A if doing so will increase $\mu(A)$), an increase of h in $\mu(A)$ implies that $\mu(A) \geq h$, as desired.

Consider the fill of a cup c swapped into A at the end of a swapping-process. Cup c's fill is at least $\mu(B) + f(n_B)$, which by (9) is at least

$$-h \cdot \frac{\delta}{1-\delta} + f(n_B) = (1-\delta)f(n_B) = h.$$

Thus the algorithm for Step 1 succeeds within $|A|$ swapping-processes, since at the end of the $|A|$-th swapping process either every cup in A has fill at least h, or the algorithm halted before $|A|$ swapping-processes because it already achieved $\mu(A) \geq h$.

After achieving $\mu(A) \geq h$, the filler performs Step 2, i.e. the filler applies alg(f) to A, and hence achieves a cup with fill at least

$$\mu(A) + f(|A|) \geq (1-\delta)f(n_B) + f(n_A),$$

as desired.

Now we analyze the running time of the filling strategy $\text{alg}(f')$. First, recall that in Step 1 $\text{alg}(f')$ calls $\text{alg}(f)$ on B, which has size n_B, as many as hn_A times. Because we mandate that $h < n$, Step 1 contributes no more than $(n \cdot n_A) \cdot T(n_B)$ to the running time. Step 2 requires applying $\text{alg}(f)$ to A, which has size n_A, once, and hence contributes $T(n_A)$ to the running time. Summing these we have

$$T'(n) \leq n \cdot n_A \cdot T(n_B) + T(n_A).$$ ◀

We next show that by recursively using the Amplification Lemma we can achieve backlog $n^{1-\varepsilon}$.

▶ **Theorem 13.** *There is an adaptive filling strategy for the variable-processor cup game on n cups that achieves backlog $\Omega(n^{1-\varepsilon})$ for any constant $\varepsilon > 0$ of our choice in running time $2^{O(\log^2 n)}$.*

Proof. Take constant $\varepsilon \in (0, 1/2)$. Let c, δ be constants that will be chosen (later) as functions of ε satisfying $c \in (0, 1), 0 < \delta \ll 1/2$. We show how to achieve backlog at least $cn^{1-\varepsilon} - 1$.

Let $\text{alg}(f_0) = \text{trivalg}$, the algorithm given by Proposition 11; recall trivalg achieves backlog $f_0(k) \geq 1/2$ for all $k \geq 2$, and $f_0(1) = 0$. Next, using the Amplification Lemma we recursively construct $\text{alg}(f_{i+1})$ as the amplification of $\text{alg}(f_i)$ for $i \geq 0$. Define a sequence g_i with

$$g_i = \begin{cases} \lceil 16/\delta \rceil, & i = 0, \\ \lfloor g_{i-1}/(1-\delta) \rfloor & i \geq 1. \end{cases}$$

We claim the following regarding this construction:

▷ Claim 14. For all $i \geq 0$,

$$f_i(k) \geq ck^{1-\varepsilon} - 1 \quad \text{for all} \quad k \in [g_i]. \tag{10}$$

Proof. We prove Claim 14 by induction on i. For $i = 0$, the base case, (10) can be made true by taking c sufficiently small; in particular, taking $c < 1$ makes (10) hold for $k = 1$, and, as $g_0 > 2$, taking c small enough to make $cg_0^{1-\varepsilon} - 1 \leq f_0(g_0) = 1/2$ makes (10) hold for $k \in [2, g_0]$ by monotonicity of $k \mapsto ck^{1-\varepsilon} - 1$[4].

As our inductive hypothesis we assume (10) for f_i; we aim to show that (10) holds for f_{i+1}. Note that, by design of g_i, if $k \leq g_{i+1}$ then $\lfloor k \cdot (1-\delta) \rfloor \leq g_i$. Consider any $k \in [g_{i+1}]$. First we deal with the trivial case where $k \leq g_0$. In this case

$$f_{i+1}(k) \geq f_i(k) \geq \cdots \geq f_0(k) \geq ck^{1-\varepsilon} - 1.$$

Now we consider the case where $k \geq g_0$. Since f_{i+1} is the amplification of f_i we have

$$f_{i+1}(k) \geq (1-\delta) f_i(\lfloor (1-\delta)k \rfloor) + f_i(\lceil \delta k \rceil).$$

By our inductive hypothesis, which applies as $\lceil \delta k \rceil \leq g_i, \lfloor k \cdot (1-\delta) \rfloor \leq g_i$, we have

$$f_{i+1}(k) \geq (1-\delta)(c \cdot \lfloor (1-\delta)k \rfloor^{1-\varepsilon} - 1) + c \lceil \delta k \rceil^{1-\varepsilon} - 1.$$

[4] Note that it is important here that ε and δ are constants, that way c is also a constant.

Dropping the floor and ceiling, incurring a -1 for dropping the floor, we have

$$f_{i+1}(k) \geq (1-\delta)(c \cdot ((1-\delta)k-1)^{1-\varepsilon} - 1) + c(\delta k)^{1-\varepsilon} - 1.$$

Because $(x-1)^{1-\varepsilon} \geq x^{1-\varepsilon} - 1$, as $x \mapsto x^{1-\varepsilon}$ is a sub-linear sub-additive function, we have

$$f_{i+1}(k) \geq (1-\delta)c \cdot (((1-\delta)k)^{1-\varepsilon} - 2) + c(\delta k)^{1-\varepsilon} - 1.$$

Moving the $ck^{1-\varepsilon}$ to the front we have

$$f_{i+1}(k) \geq ck^{1-\varepsilon} \cdot \left((1-\delta)^{2-\varepsilon} + \delta^{1-\varepsilon} - \frac{2(1-\delta)}{k^{1-\varepsilon}}\right) - 1.$$

Because $(1-\delta)^{2-\varepsilon} \geq 1 - (2-\varepsilon)\delta$, a fact called Bernoulli's Identity, we have

$$f_{i+1}(k) \geq ck^{1-\varepsilon} \cdot \left(1 - (2-\varepsilon)\delta + \delta^{1-\varepsilon} - \frac{2(1-\delta)}{k^{1-\varepsilon}}\right) - 1.$$

Of course $-2(1-\delta) \geq -2$, so

$$f_{i+1}(k) \geq ck^{1-\varepsilon} \cdot \left(1 - (2-\varepsilon)\delta + \delta^{1-\varepsilon} - \frac{2}{k^{1-\varepsilon}}\right) - 1.$$

Because

$$\frac{-2}{k^{1-\varepsilon}} \geq \frac{-2}{g_0^{1-\varepsilon}} \geq -2(\delta/16)^{1-\varepsilon} \geq -\delta^{1-\varepsilon}/2,$$

which follows from our choice of $g_0 = \lceil 16/\delta \rceil$ and the restriction $\varepsilon < 1/2$, we have

$$f_{i+1}(k) \geq ck^{1-\varepsilon} \cdot \left(1 - (2-\varepsilon)\delta + \delta^{1-\varepsilon} - \delta^{1-\varepsilon}/2\right) - 1.$$

Finally, combining terms we have

$$f_{i+1}(k) \geq ck^{1-\varepsilon} \cdot \left(1 - (2-\varepsilon)\delta + \delta^{1-\varepsilon}/2\right) - 1.$$

Because $\delta^{1-\varepsilon}$ dominates δ for sufficiently small δ, there is a choice of $\delta = \Theta(1)$ such that

$$1 - (2-\varepsilon)\delta + \delta^{1-\varepsilon}/2 \geq 1.$$

Taking δ to be this small we have,

$$f_{i+1}(k) \geq ck^{1-\varepsilon} - 1,$$

completing the proof. We remark that the choices of c, δ are the same for every i in the inductive proof, and depend only on ε. ◁

To complete the proof, we will show that g_i grows exponentially in i. This implies that there exists $i_* \leq O(\log n)$ such that $g_{i_*} \geq n$, and hence we have an algorithm $\mathrm{alg}(f_{i_*})$ that achieves backlog $cn^{1-\varepsilon} - 1$ on n cups, as desired.

We lower bound the sequence g_i with another sequence g'_i defined as

$$g'_i = \begin{cases} 4/\delta, & i = 0 \\ g'_{i-1}/(1-\delta) - 1, & i > 0. \end{cases}$$

Solving this recurrence, we find

$$g'_i = \frac{4-(1-\delta)^2}{\delta}\frac{1}{(1-\delta)^i} \geq \frac{1}{(1-\delta)^i},$$

which clearly exhibits exponential growth. In particular, let $i_* = \left\lceil \log_{1/(1-\delta)} n \right\rceil$. Then, $g_{i_*} \geq g'_{i_*} \geq n$, as desired.

Let the running time of $f_i(n)$ be $T_i(n)$. From the Amplification Lemma we have following recurrence bounding $T_i(n)$:

$$\begin{aligned} T_i(n) &\leq n\lceil \delta n \rceil \cdot T_{i-1}(\lfloor (1-\delta)n \rfloor) + T_{i-1}(\lceil \delta n \rceil) \\ &\leq 2n^2 T_{i-1}(\lfloor (1-\delta)n \rfloor). \end{aligned}$$

It follows that $\text{alg}(f_{i_*})$, recalling that $i_* \leq O(\log n)$, has running time

$$T_{i_*}(n) \leq (2n^2)^{O(\log n)} \leq 2^{O(\log^2 n)},$$

as desired. ◀

Now we provide a construction that can achieve backlog $\Omega(n)$ in very long games. The construction can be interpreted as the same argument as in Theorem 13 but with an extremal setting of δ to $\Theta(1/n)$[5].

▶ **Theorem 15.** *There is an adaptive filling strategy that achieves backlog $\Omega(n)$ in time $O(n!)$.*

Proof. First we construct a slightly stronger version of trivalg that achieves backlog 1 on $n \geq n_0 = 8$ cups, instead of just backlog $1/2$; this simplifies the analysis.

▷ Claim 16. There is a filling algorithm trivalg_2 that achieves backlog at least 1 on $n_0 = 8$ cups.

Proof. Let trivalg_1 be the amplification of trivalg using $\delta = 1/2$. On 4 cups trivalg_1 achieves backlog at least $(1/2)(1/2) + 1/2 = 3/4$. Let trivalg_2 be the amplification of trivalg_1 using $\delta = 1/2$. On 8 cups trivalg_2 achieves backlog at least $(1/2)(3/4) + 3/4 \geq 1$. ◁

Let $\text{alg}(f_0) = \text{trivalg}_2$; we have $f_0(k) \geq 1$ for all $k \geq n_0$. For $i > 0$ we construct $\text{alg}(f_i)$ as the amplification of $\text{alg}(f_{i-1})$ using the Amplification Lemma with parameter $\delta = 1/(i+1)$.

We claim the following regarding this construction:

▷ Claim 17. For all $i \geq 0$,

$$f_i((i+1)n_0) \geq \sum_{j=0}^{i} \left(1 - \frac{j}{i+1}\right). \tag{11}$$

Proof. We prove Claim 17 by induction on i. When $i = 0$, the base case, (11) becomes $f_0(n_0) \geq 1$ which is true. Assuming (11) for f_{i-1}, we now show (11) holds for f_i. Because f_i is the amplification of f_{i-1} with $\delta = 1/(i+1)$, we have by the Amplification Lemma

$$f_i((i+1)\cdot n_0) \geq \left(1 - \frac{1}{i+1}\right) f_{i-1}(i \cdot n_0) + f_{i-1}(n_0).$$

[5] Or more precisely, setting δ in each level of recursion to be $\Theta(1/n)$, where n is the subproblem size; note in particular that δ changes between levels of recursion, which was not the case in the proof of Theorem 13.

Since $f_{i-1}(n_0) \geq f_0(n_0) \geq 1$ we have

$$f_i((i+1)\cdot n_0) \geq \left(1 - \frac{1}{i+1}\right) f_{i-1}(i \cdot n_0) + 1.$$

Using the inductive hypothesis we have

$$f_i((i+1)\cdot n_0) \geq \left(1 - \frac{1}{i+1}\right) \sum_{j=0}^{i-1} \left(1 - \frac{j}{i}\right) + 1.$$

Note that

$$\left(1 - \frac{1}{i+1}\right) \cdot \left(1 - \frac{j}{i}\right) = \frac{i}{i+1} \cdot \frac{i-j}{i} = \frac{i-j}{i+1} = 1 - \frac{j+1}{i+1}.$$

Thus we have the desired bound:

$$f_i((i+1)\cdot n_0) \geq \sum_{j=1}^{i} \left(1 - \frac{j}{i+1}\right) + 1 = \sum_{j=0}^{i} \left(1 - \frac{j}{i+1}\right). \qquad \lhd$$

Let $i_* = \lfloor n/n_0 \rfloor - 1$, which by design satisfies $(i_*+1)n_0 \leq n$. By Claim 17 we have

$$f_{i_*}((i_*+1)\cdot n_0) \geq \sum_{j=0}^{i_*} \left(1 - \frac{j}{i_*+1}\right) = i_*/2 + 1.$$

As $i_* = \Theta(n)$, we have thus shown that $\mathrm{alg}(f_{i_*})$ can achieve backlog $\Omega(n)$ on n cups.

Let T_i be the running time of $\mathrm{alg}(f_i)$. The recurrence for the running running time of f_{i_*} is

$$T_i(n) \leq n \cdot n_0 T_{i-1}(n - n_0) + O(1).$$

Clearly $T_{i_*}(n) \leq O(n!)$. ◀

References

1. Micah Adler, Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul Goldberg, and Mike Paterson. A proportionate fair scheduling rule with good worst-case performance. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 101–108, 2003. doi:10.1145/777412.777430.
2. Amihood Amir, Martin Farach, Ramana M. Idury, Johannes A. La Poutré, and Alejandro A. Schäffer. Improved dynamic dictionary matching. *Inf. Comput.*, 119(2):258–282, 1995. doi:10.1006/inco.1995.1090.
3. Amihood Amir, Gianni Franceschini, Roberto Grossi, Tsvi Kopelowitz, Moshe Lewenstein, and Noa Lewenstein. Managing unbounded-length keys in comparison-driven data structures with applications to online indexing. *SIAM Journal on Computing*, 43(4):1396–1416, 2014.
4. Yossi Azar and Arik Litichevskey. Maximizing throughput in multi-queue switches. *Algorithmica*, 45(1):69–90, 2006.
5. Amotz Bar-Noy, Ari Freund, Shimon Landa, and Joseph (Seffi) Naor. Competitive on-line switching policies. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 525–534, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545452.
6. Amotz Bar-Noy, Aviv Nisgav, and Boaz Patt-Shamir. Nearly optimal perfectly periodic schedules. *Distributed Computing*, 15(4):207–220, 2002.

7 S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996. doi:10.1007/BF01940883.

8 Sanjoy K Baruah, Johannes E Gehrke, and C Greg Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the 9th International Parallel Processing Symposium*, pages 280–288, 1995.

9 Michael Bender, Rathish Das, Martín Farach-Colton, Rob Johnson, and William Kuszmaul. Flushing without cascades. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.

10 Michael Bender, Martín Farach-Colton, and William Kuszmaul. Achieving optimal backlog in multi-processor cup games. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, 2019.

11 Michael Bender and William Kuszmaul. Randomized cup game algorithms against strong adversaries. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2021.

12 Michael A Bender, Rezaul A Chowdhury, Rathish Das, Rob Johnson, William Kuszmaul, Andrea Lincoln, Quanquan C Liu, Jayson Lynch, and Helen Xu. Closing the gap between cache-oblivious and cache-adaptive analysis. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 63–73, 2020.

13 Michael A Bender, Rezaul A Chowdhury, Rathish Das, Rob Johnson, William Kuszmaul, Andrea Lincoln, Quanquan C Liu, Jayson Lynch, and Helen Xu. Closing the gap between cache-oblivious and cache-adaptive analysis. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 63–73, 2020.

14 Michael A Bender, Roozbeh Ebrahimi, Jeremy T Fineman, Golnaz Ghasemiesfeh, Rob Johnson, and Samuel McCauley. Cache-adaptive algorithms. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 958–971. SIAM, 2014.

15 Peter Damaschke and Zhen Zhou. On queuing lengths in on-line switching. *Theoretical computer science*, 339(2-3):333–343, 2005.

16 Paul Dietz and Daniel Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 365–372, 1987. doi:10.1145/28395.28434.

17 Paul F. Dietz and Rajeev Raman. Persistence, amortization and randomization. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 78–88, 1991. URL: http://dl.acm.org/citation.cfm?id=127787.127809.

18 Johannes Fischer and Paweł Gawrychowski. Alphabet-dependent string searching with wexponential search trees. In *Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 160–171, 2015.

19 Rudolf Fleischer and Hisashi Koga. Balanced scheduling toward loss-free packet queuing and delay fairness. *Algorithmica*, 38(2):363–376, February 2004. doi:10.1007/s00453-003-1064-z.

20 H Richard Gail, G Grover, Roch Guérin, Sidney L Hantler, Zvi Rosberg, and Moshe Sidi. Buffer size requirements under longest queue first. *Performance Evaluation*, 18(2):133–140, 1993.

21 Leszek Gasieniec, Ralf Klasing, Christos Levcopoulos, Andrzej Lingas, Jie Min, and Tomasz Radzik. Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 229–240. Springer, 2017.

22 Michael H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, 2010. doi:10.1145/1753171.1753195.

23 Michael T Goodrich and Paweł Pszona. Streamed graph drawing and the file maintenance problem. In *International Symposium on Graph Drawing*, pages 256–267. Springer, 2013.

24 Nan Guan and Wang Yi. Fixed-priority multiprocessor scheduling: Critical instant, response time and utilization bound. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 2470–2473. IEEE, 2012.

25 Tsvi Kopelowitz. On-line indexing for general alphabets via predecessor queries on subsets of an ordered list. In *Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 283–292, 2012.

26 William Kuszmaul. Achieving optimal backlog in the vanilla multi-processor cup game. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.

27 Andrea Lincoln, Quanquan C Liu, Jayson Lynch, and Helen Xu. Cache-adaptive exploration: Experimental results and scan-hiding for adaptivity. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, pages 213–222, 2018.

28 Ami Litman and Shiri Moran-Schein. On distributed smooth scheduling. In *Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 76–85, 2005.

29 Ami Litman and Shiri Moran-Schein. Smooth scheduling under variable rates or the analog-digital confinement game. *Theor. Comp. Sys.*, 45(2):325–354, June 2009. doi:10.1007/s00224-008-9134-x.

30 Ami Litman and Shiri Moran-Schein. On centralized smooth scheduling. *Algorithmica*, 60(2):464–480, 2011.

31 Chung Laung Liu. Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Programs Summary, 1969*, 1969.

32 Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

33 Richard T Mills, Andreas Stathopoulos, and Dimitrios S Nikolopoulos. Adapting to memory pressure from within scientific applications on multiprogrammed cows. In *Proc. 8th International Parallel and Distributed Processing Symposium (IPDPS)*, page 71, 2004.

34 Mark Moir and Srikanth Ramamurthy. Pfair scheduling of fixed and migrating periodic tasks on multiple resources. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 294–303, 1999.

35 Christian Worm Mortensen. Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 618–627, 2003.

36 Michael Rosenblum, Michel X Goemans, and Vahid Tarokh. Universal bounds on buffer size for packetizing fluid policies in input queued, crossbar switches. In *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 2, pages 1126–1134, 2004.