An investigation of variable-processor cup games

Alek Westover

March 2, 2020

In the ***variable-processor cup game*** the filler is allowed to change p, the amount of water that the filler can add and the number of cups from which the emptier can remove water. This is a natural extention of the multi-processor cup game. Note that although the restriction that the filler and emptier's resources vary together may seem artificial, this is the only way to conduct the analysis; the rationale behind giving the emptier and filler equal resources in the classical vanilla multi-processor cup game is that this is the only way to achieve upper and lower bounds. The equivalent rational holds for the motivation of the variable-processor cup game. Analysis of this game does provide information about how real-world systems will behave.

Apriori the fact that the number of processors can vary offers neither the filler nor the emptier a clear advantage: lower values of p mean that the emptier is at more of a discretization disadvantage but also mean that the filler can anchor fewer cups. We hoped that the variable-processor cup game could be simulated in the vanilla multiprocessor cup game, because the extra ability given to the filler does not seem very strong. The new version of the cup game arose as we tried to get a bound of $\Omega(\log p)$ backlog in the multiprocessor game against an off-line filler, which would combine with previous results to give us a lower bound that matches our upper bound: $O(\log\log n + \log p)$. This new version seemed promising in this respect because of the following Proposition:

Proposition 1. *In the variable-processor cup game on n cups with an off-line filler attacking a smoothed-greedy opponent, the filler can force backlog to be $\Omega(\log n)$.* [1]

However, we show that attempts at simulating the variable-processor cup game are futile because the variable-processor cup game is–surprisingly–fundamentally different from the multiprocessor cup game, and thus impossible to simulate. Formally this is a corollary of the ***Amplification Lemma***:

Lemma 1. *Given a filling algorithm for achieving fill $f(k)$ a cup in the cup game on k cups relative to a baseline of the average fill of the cups, with* ***negative fill****–fill below the average fill–allowed, there exists a filling algorithm for achieving "amplified" fill of at least*

$$f'(k) = \frac{1}{2}(f(k/2) + f(k/4) + \cdots)$$

in one of the k cups.

We proceed to prove these claims.

Before proving Proposition 1, we prove the following useful proposition concerning an important subroutine that the filler will use.

Proposition 2. *In the variable-processor cup game on n cups against an offline filler where the filler does not increase the number of processors above n, there exists a filling strategy that guarantees–with probability at least $1 - e^{O(-n)}$–either to result in $\Theta(n)$ cups with fill $\Omega(1)$, or to result in some cup with fill $\tilde{\Omega}(n)$, where fill is relative to average fill, with negative fill allowed.*

there is another case: gets a good number of anchor cups. Q: how do we USE this other case in prop2?

[1] Note that we have $\Omega(\log n)$ in this proposition instead of $\Omega(\log p)$ because the filler can increase the number of processors, so it increases the number of processors to $n-1$ to start. A nearly ientical construction could be used to show that backlog $\Omega(\log p_{\max})$ can be achieved, where the number of processors starts at $p_{\max}$ and the filler does not ever increase the number of processors. However, using $p_{\max} = n$ is natural in the variable-processor cup game, so we do not consider the game with the restriction that the filler can not increase the number of processors above some $p_{\max} < n$.

Proof. A problem that arises in trying to prove that we can achieve $\Theta(n)$ cups with fill $\Omega(1)$ is that many cups, in the most extreme case all but a single cup, could have negative fill (fill below the average relative to which fill is measured). However, if the fill is so unevenly distributed then this means that the backlog is already very high, so the filler does to achieve $\Theta(n)$ cups with fill $\Omega(1)$ in the context to which this proposition must be applied. Furthermore, there do exist emptying algorithms that can act in such a way as to disrupt our strategy for achieving $\Theta(n)$ cups with backlog $\Omega(1)$, but we shall show that such emptiers incur enormous backlog; in particular backlog $\tilde{\Omega}(n)$.

Let $h \geq \Omega(1)$ be the backlog we aim to achieve in $\Theta(n)$ cups. If there exists a $k < n$ such that the k fullest cups have fill that sums to more than $h \cdot n$, then consider the least k, $k_{\min}$, such that this is true. We call the $k_{\min}$ fullest cups ***overpowered***, because they are pushing the average fill of the other cups down significantly. Note that overpowered cups must have positive fill; otherwise an overpowered cup could be removed while maintaining the requirement for the sum of their fills, which contradicts the minimality of $k_{\min}$.

If there is a cup with fill at least $h \cdot n/\text{loglog}n \geq \tilde{\Omega}(n)$, then we don't need to worry about attaining $\Theta(n)$ cups with fill at least h; the proposition guarantees that we either attain those cups, or attain one cup with enormous fill. Otherwise, the number of overpowered cups cannot be in $\{1,2,...,\text{loglog}n\}$.

Now we consider the case where there are at least $\text{loglog}n$ overpowered cups, but no one cup has fill larger than $h \cdot n/\text{loglog}n$. The algorithm we will describe forms a set of $n/2$ ***anchor*** cups, of which $k \cdot n$, for some appropriate $k < \frac{1}{2}$ (k is determined by h, the larger h is desired the smaller k will be) will have fill at least h, and the algorithm will have success probability at least $1 - e^{-nk/10}$. The algorithm is completely symmetric; every cup has the same probability of being part of the $n/2$ anchor cups: $1/2$. By a Chernoff Bound the probability that less than $(\text{loglog}n)/4$ overpowered cups end up in the anchor set is less than $1/\text{log}n$.

Now we consider the case in which there are no overpowered cups. The filler anchors $n/2$ cups and repeats the following algorithm for each anchored cup i:
At every step the filler adds one unit of fill to each anchored cup. For $(n/2)^c$ (for some constant $c > 2$, e.g. $c = 3$) rounds the filler plays a single processor cup game on $s \geq e^h$ non-anchored cups (e.g. $s = 21$) that lasts for $s-1$ steps. The filler choses a new set of $s-1$ cups for each of these single processor cup games to guarantee that the cups do not have "negative fill" (fill below the average fill) to start The filler's strategy will strongly disincentivise the emptier from not emptying from each of the $n/2$ anchor cups most time; for simplicities sake we first analyze the case in which the emptier does empty from multiple cups outside the anchor set ever. Over the $s-1$ steps the emptier will remove from $s-1$ cups. In the worst case these are all distinct members of the $s-1$ cups. We can predict this sequence with constant probability, specifically probability $1/s!$. Hence with probability at least $1/s!$ we can achieve fill at least $\frac{1}{s} + \frac{1}{s-1} + \cdots + \frac{1}{1} > \ln s \geq h$ in this cup.

For each for each of the anchor cups the filler choses a round uniformly at random from the $(n/2)^c$ rounds to swap the cup that has constant probability of having at least constant fill with the anchor cup being focussed on at that point.

Thus, at the end of this process each cup in the anchor set has at least a $1/s!$ chance of having fill at least h. The expectation of the number of cups that have fill at least h is at least $(n/2)/s!$. Using a Chernoff bound we can show that with exponentially high probability in n the actual number of such cups does not deviate from this mean by more than a constant factor. In particular, let m_A be the actual number of cups with fill at least h in the anchor set, and let m_E be the expected number (i.e. $m_E = (n/2)/s!$) of such cups in the anchor set. Then

$$\Pr[m_A < m_E/2] \leq e^{-n/(40 \cdot s!)}.$$

Now we consider the case where the emptier choses some rounds to empty from multiple non-anchor cups. This is problematic since it could mean that we are not genuinely playing a single processor cup game on the selected 21 cups. However, neglecting an anchor cups is dangerous for the emptier, as it will substantially increase the fill in the anchor set. If the emptier neglects the anchor set in d of the p^c rounds, then the probability of the emptier interfering with the single processor cup game on the 1 round that the filler has selected to swap the cup into the anchor set is d/p^c. If this is small, e.g. less than $1/2$, then we can simply accept a small constant reductive factor on the number of cups that we are guaranteeing have the desired constant fill. On the other hand, if $d/p^c > 1/2$, then $d \geq \Omega(p^c)$. This means that the average fill in the anchor set has increased dramatically, by poly(p). Thus at least one of the anchor cups must have fill $\Omega(\text{poly}(p))$. This guarantees fill $\Omega(\text{poly}p)$ for $\Omega(\text{poly}(p))$ rounds, which is sufficient to satisfy the proposition. □

Proof of Proposition 1. The filler follows the following algorithm:

1. Apply the subroutine described in Proposition 2 to attain either a cup with incredibly large fill ($\Omega(\text{poly}(n))$), or $n \cdot g = \Theta(n)$ cups with fill $h \geq \Omega(1)$ (with probability at least $1 - e^{O(-n)}$).

2. Decrease the number of processors to $p' = n \cdot g$.

3. Over the next $\lfloor h/2 \rfloor - 1$ steps the filler places 1 water in each of p' cups. Assuming that the emptier is using the smoothed-greedy algorithm, then this will result in the p' chosen–*known*–cups having fill at least $\lfloor h/2 \rfloor - 1$.

4. Recurse on the known cups.

If at any point in the process backlog $\Omega(\text{poly}(n))$ was achieved then the emptier succeeds by definition, having achieved very high backlog.

Otherwise, Proposition 2 guaratnees that at each recursive step the backlog will increase by at least $\lfloor h/2 \rfloor - 1$. We can chose h such that this is e.g. 1.

By recursing $\Theta(\log n)$ times, increasing backlog by at least 1 at each recursion level, we get backlog of $\Omega(\log n)$ as desired. In order for this guarantee to hold with good probability we do not recurse until the number of processors has been reduced to 1; Rather we cut off at halfway through the recursion, i.e. when the number of processors has been reduced to $\Theta(\sqrt{n})$. This gives us the same backlog (asymptotically), while also making the probability of failure to achieve this backlog less than $e^{O(-\sqrt{n})}\log n$ by a union bound on the exponentially small failure probability at each recursive step. $\square$

Next we establish the Amplification Lemma:

Proof of Lemma 1. If, at any point in the process that will be described, backlog is greater than $f'(k)$, then the filler stops and the Lemma is satisfied as the desired backlog having been achieved.

The main idea of this analysis is to use f to achieve fill $f(n/2^l)$ on half of the ***active*** cups (of which there are $n/2^l$), and then halving the number of processors, and the number of cups that the filler is focussed on, and recursing on this set.

A key technical challenge is handling ***negative fill***: fill below the average fill of the active cups. Note that it is strictly easier for the filler to achieve large fill if negative fill is not allowed (i.e. cups can zero out); we need to allow for negative fill because we are recursing.

Let $h_l = \frac{1}{2} f(k/2^l)$; the filler will achieve fill at least $h_l/4$ in $n_l = n/2^l$ cups at the l-th level of recursion. Let a cup be designated ***bad*** if it has fill less than $-h_l/4$, ***good*** if it has fill at least $h_l/4$ and ***fine*** if it has fill in $(-h_l/4, h_l/4)$. Let b be the number of bad cups, g the number of good cups, and f the number of fine cups.

If $g \geq n_l/4$ then we simply recurse on the good cups. wait noo

If $f \geq n_l/4$. Which is fine. So we do the following, with the assurance that the cups are pretty much fine basically.

The filler anchors $n/2$ cups. Then, the filler applies the algorithm to achieve backlog $f(n/2)$ on the $n/2$ non-anchored cups, and swaps the cups into the anchor set upon achieving this backlog. Note that this backlog is relative to the average fill in those cups, which is depressed by the process of siphoning water out of the non-anchored cups into the anchor set. In particular, say that the initial average fill was μ, and that the final average fill in the non-anchor set is μ'. The average fill in the anchor set is at least $\mu' + f(n/2)$. Hence the average fill of all the cups is at least $\mu' + f(n/2)/2$. But the average fill at the end of the process is the same as the average fill at the start of the process, so $\mu \geq \mu' + f(n/2)/2$. The fact that the non-anchor cups have sunk by at least $f(n/2)/2$ implies that the anchor cups must have risen by at least this amount (on average) to maintain the average fill. Note that f assumes negative water will be incurred for cups that go below the average, however in the real game there is a hard threshold at 0, below which there is no negative water. This is not a problem; in fact it is benneficial for the filler! We can easily slightly revise the above argument to accomodate the fact that cups can zero out: Let the final average fill be $\mu + \epsilon$ for some $\epsilon > 0$. We have $\mu + \epsilon \geq \mu' + f(n/2)/2$, so the average in the anchor cups must be at least $\mu + \epsilon + f(n/2)/2$ to achieve the correct average fill.

Upon achieving fill $\frac{1}{2} f(n/2)$ in the $n/2$ anchor cups, the filler will cut the number of processors in half, and proceed to focus only on the cups that formed the anchor set. By recursing with an identical method the filler can get backlog $\frac{1}{2}(f(n/2) + f(n/4) + f(n/8) + \cdots)$.

Note that the ability to increase the number of processors is needed in the recursive calling of f, as f might require using lower values of p, but at the end of performing f we must restore the number of processors back up to the number that we started with so that the filler can continue with the same value of p that it started with. $\square$

We now use the Amplication Lemma to achieve the following:

Corollary 1. *The filler can achieve backlog $\Omega(poly(n))$*

Proof. We recursively construct functions f_m by application of the Amplification Lemma. We will start with

$$f_0(k) = \begin{cases} \log_2 k, & k \geq 1, \\ 0 & \text{else.} \end{cases}$$

We then construct f_{m+1} as the ***amplification***of f_m. By repeated application of this procedure $\log_2 n^{1/9}$ times we achieve a function $f_{\log_2 \sqrt{n}}(k)$ with the property that for $k \geq n$, $f_{\log_2 n^{1/9}}(k) \geq 2^{\log_2 n^{1/9}} \log_2 k$. In particular, this gives a filling strategy that when applied to n cups gives backlog $\Omega(n^{1/9}\log_2 n) \geq \Omega(\text{poly}(n))$ as desired. To prove this, we prove the following lowerbound for f_m by induction:

$$f_m(k) \geq 2^m \log_2 k, \text{ for } k \geq (2^9)^m.$$

The base case follows from the definition of f_0. Assuming the property for f_m, we get the following:

$$\begin{aligned} \text{for } k > (2^9)^{m+1}, f_{m+1}(k) &= \frac{1}{2}(f_m(k/2) + f_m(k/4) + \cdots + f_m(k/2^9) + \cdots) & (1)\\ &\geq \frac{1}{2}(f_m(k/2) + f_m(k/4) + \cdots + f_m(k/2^9)) & (2)\\ &\geq \frac{1}{2}2^m(\log_2(k/2) + \log_2(k/4) + \cdots + \log_2(k/2^9)) & (3)\\ &\geq \frac{1}{2}2^m(9\log_2(k) - \frac{9 \cdot 10}{2}) & (4)\\ &\geq 2^{m+1}\log_2(k) & (5) \end{aligned}$$

as desired. Hence the inductive claim holds, which establishes that $f_{\log_2 n^{1/9}}$ satisfies the desried condition, which proves that backlog can be made $\tilde{\Omega}(n^{1/9})$.

Generalizing this approach we can achieve a slightly better polynomial lowerbound on backlog. In our construction the point after which we had a bound for f_m grew further out by a factor of 2^9 each time. Instead of 2^9 we now use 2^α for some $\alpha \in \mathbb{N}$, and can find a better value of α. The value of α dictates how many itterations we can perform: we can perform $\log_2 n^{1/\alpha}$ itterations. The parameter α also dictates the multiplicative factor that we gain upon going from f_m to f_{m+1}. For $\alpha = 9$ this was 2. In general it turns out to be $\frac{\alpha-1}{4}$. Hence, we can achieve backlog $\Omega\left(\left(\frac{\alpha-1}{4}\right)^{\log_2 n^{1/\alpha}} \log_2 n\right)$. This optimizes at $\alpha = 13$, to backlog $\Omega(n^{\frac{\log_2 3}{13}} \log n) \approx \Omega(n^{0.122}\log n)$. We can actually even improve over this slightly. Note that in the proof that f_{m+1} gains a factor of 2 over f_m given above, the transition from (4) to (5) is usually very loose: for small m a significant portion of the $\log_2 k$ is annihlated by the constant $1+2+\cdots+9$ (or in general $1+2+\cdots+\alpha$), but for larger values of m because k must be large we can get larger factors between steps, in theory factors arbitrarily close to $\alpha/2$. If we could gain a factor of $\alpha/2$ at each step, then the backlog achievable would be $\Omega((\alpha/2)^{\log_2 n^{1/\alpha}} \log n) = \Omega(n^{(\log_2 \alpha/2)/2} \log n)$ which optimizes (over the naturals) at $\alpha = 5$ to $n^{(\log_2 5/2)/5} \approx n^{0.264}$. However, we can't actually gain a factor of $\alpha/2$ each time because of the subtracted constant. But, for any $\epsilon > 0$ we can achieve a $\alpha/2 - \epsilon$ factor increase each time (for sufficiently large m). Of course ϵ can't be made arbitrarily small becasue m can't be made arbitrarily large, and the "cut off" m where we start achieving the $\alpha/2 - \epsilon$ factor increase must be a constant (not dependent on n). When the cutoff m, or equivalently ϵ, is constant then we can achieve backlog $\Omega((\alpha/2 - \epsilon)^{\log_2 n^{1/\alpha}} \log n) = \Omega(n^{(\log_2 \alpha/2 - \epsilon)/2} \log n)$. For instance, with this method we can get backlog $\tilde{\Omega}(n^{1/4})$ for appropriate ϵ, α choice, or $\tilde{\Omega}(n^{(\log_2 5/2 - \epsilon)/5})$ for any constant $\epsilon > 0$. We could potentially aim to achieve even higher backlog by using more than the first α terms of the sum. The terms after $f_m(k/2^\alpha)$ in the sum are evaluated at points where they are potentially positive, but will not have the full strength of the $2^m \log_2 k$. This makes them difficult to deal with, and as it is not likely that we will get anything besides a modest increase in the exponent of our polynomial we do not pursue this.

□

Remark 1. *The recursive construction requires quite a lot of steps, in fact a superpolynomial number of steps. If we consider the tree that represents computation of $f_{\log n}(n)$ we see that each node will have at most α (some constant, e.g. $\alpha = 9$, α is the number of terms that we keep in the sum) children (the children of $f_k(c)$ are*

$f_{k-1}(c/2)$,$f_{k-1}(c/4)$,...$f_{k-1}(c/2^\alpha)$), and the depth of the tree is $\log n^{1/\alpha}$. Say that the running time at the node $f_{\log n}(n)$ is $T(n)$. Then because $f_k(n)$ must call each of $f_{k-1}(n/2^i)$ $n/2^i$ times for $1 \le i \le \alpha$, we have that $T(n) \le \frac{\alpha n}{2} T(n/2)$. This recurrence yeilds $T(n) \le poly(n)^{\log n} = O(2^{\log^2 n})$ for the running time.

Becasue of the large backlog achievable by the on-line filler in the variable-processor cup game, and the fact that the off-line filler can achieve a set of cups with known constant fill with failure probability exponentially small in the variable-processor cup game, we conjectured that Proposition 1 can be improved in a manner similar to how the on-line filling strategy was improved. We adapt the Amplification Lemma to the case of an oblivious filler, and achieve as a corollary the desired higher backlog. The ***Oblivious Amplification Lemma:***

Lemma 2. *Given an oblivious filling algorithm for achieving a cup with fill at least $f(k)$ in the variable-processor cup game on k cups (this backlog is relative to a baseline of the average of the cups, with "negative fill"–fill below the average–allowed even in the initial state) that succeeds with probability at least $1/2$, there exists an oblivious filling algorithm for achieving "amplified" fill of*

$$f'(k) = \frac{1}{2}(f(k/2) + f(k/4) + \cdots + f(k/2^9))$$

in a cup with probability at least $1/2$.

Proof. This is a simple application of a Chernoff bound. The filler partitions the cups into 2 groups of size $k/2$ and then anchors half of them and repeatedly applies $f(k)$ on the non-anchored cups. The filler has to apply $f(k)$ a lot of times, poly(k), and say "psych" all but one of these times, this time being chosen at random, and in that time it swaps the cup into the anchor set. This guarantees that it is a bad idea for the emptier to not be focussing on the anchor cups. Thus at the end of this process each cup in the anchor set will have at least a $1/2$ chance of hainvg fill $f(k/2)/2$ fill in it. At this point, assuming that the opponent is smoothed greedy, the filler can transfer the water into known cups, and then recurse. In order to achive a failure probability of at most $1/2$, the filler doesn't recurse all the way to the bottom. Rather, it stops at some level β. As each level of recursion fails with probability at most $e^{-k/10}$, the probability that any level fails is bounded above, by the union bound, by $e^{-n/10} + e^{-n/(2\cdot 10)} + e^{-n/(4\cdot 10)} + \cdots e^{-n/(2^\beta \cdot 10)}$ This sum is bounded above by $e^{-n/(2^\beta \cdot 10)} \log n$, so as long as β is not too huge this shouldn't be an issue to get to be less than $1/2$. □

Corollary 2. *There exists an off-line filling strategy that achieves backlog $\Omega(2^{\sqrt{\log n}/2})$ with high probability in time $O(n)$.*

With the aid of Lemma 2 we can now get our backlog guarantee, much as in Corollary 1.

Proof. First cut n to $n' = 2^{\sqrt{\log n}}$. Note that $\log n' = \sqrt{\log n}$ and $\log^2 n' = \log n$, so by Remark 1 we would expect the running time of this algorithm to be $2^{\log^2 n'} \le O(n)$, and we would expect the backlog obtained to be $2^{\log n'} = 2^{\sqrt{n}}$. We will now show that this is pretty much the case. And this will all happen with high probability. Basically, just apply Lemma 2 $\log n'$ times. Yay, backlog poly(n'). Note that it is in the advertised running time. Double yay. □