An investigation of variable-processor cup games

Alek Westover

February 26, 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. Apriori this offers neither the filler nor the emptier a clear advantage: lower values of p mean that the emptier is at more of a discretization advantage 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 out upper bound: $O(\text{loglog}n+\log p)$. This new version seemed promising in this respect because of the following Lemma:

Lemma 1. *In the variable-processor cup game on n cups against an off-line filler, the filler can force backlog to be $\Omega(\log p)$.*

However, attempts at simulating the variable-processor cup game are futile because of the following Lemmas, which shows that the variable-processor cup game is, surprisingly, fundamentally different than the multiprocessor cup game, and thus impossible to simulate:

Lemma 2. *In the variable-processor cup game on n cups against an online filler, the filler can force backlog to be $\Omega(\log^2 p)$.*

In fact Lemma 2 gives a fairly weak lowerbound on backlog; the filler can achieve much higher fill, as high as $\text{poly}(n)$, which follows as a corollary of the ***Amplification Lemma:***

Lemma 3. *Given a filling algorithm for achieving $f(k)$ backlog 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), there exists a filling algorithm for achieving "amplified" backlog of*

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

on k cups.

We proceed to prove the Lemmas.

Before proving Lemma 1, we prove the following useful proposition concerning a subroutine that the filler will use in the Lemma, and a later result.

Proposition 1. *There exists a filling strategy that guarantees–against a smoothed-greedy opponent– that $\Theta(p)$ cups will have fill at least* $\ln 21$ *(i.e. large consant fill) with failure probability at most $O(e^{-p})$.*

Proof. The filler anchors $p-1$ 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 p^c (for some constant $c>2$) rounds the filler plays a single processor cup game on 21 non-anchored cups that lasts for 20 steps. The filler choses a new set of 20 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 $p-1$ 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 20 steps the emptier will remove from 20 cups. In the worst case these are all distinct members of the 21 cups. We can predict this sequence with constant probability, specifically probability $1/21!$. Hence with probability at least $1/21!$ we can achieve fill at least $\frac{1}{21}+\frac{1}{19}+\cdots+\frac{1}{1}>\ln 21$ in this cup.

For each for each of the anchor cups the filler choses a round uniformly at random from the p^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 $1/21!$ chance of having fill at least ln21. The expectation of the number of cups that have fill at least ln21 is at least $(p-1)/21!$. Using a Chernoff bound we can show that with high probability in p 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 ln21 in the anchor set, and let m_E be the expected number (i.e. $m_E=(p-1)/21!$) of such cups in the anchor set. Then

$$\Pr[m_A < m_E/2] \leq e^{-p/(12\cdot 21!)}.$$

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 $\text{poly}(p)$. 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. But this could never happen against smoothed-greedy!

□

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

1. Get $(p-1)/(21!\cdot 2)$ cups with fill at least ln21 with probability at least $1-e^{-p/(12\cdot 21!)}$ (possible by Proposition 1)
2. Decrease the number of processors from p to $p'=(p-1)/(21!\cdot 2)$. Over the next $\lfloor((\ln 21)-1)/2\rfloor=1$ steps the filler places 1 water in each of p' cups. Assuming that the emptier is using smoothed-greedy then this will result in having fill $\lfloor((\ln 21)-1)/2\rfloor=1$ in these *known* cups.
3. Recurse on those known cups.

By recursing $\Theta(\log p)$ times, increasing backlog by at least 1 at each recursion level, we get backlog of $\Omega(\log p)$ 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 $\Theta(\sqrt{p})$. This gives us the same backlog (asymptotically), while also making the probability of failure to achieve this backlog $\tilde{O}(e^{-p})$ by a union bound on the exponentially small failure probability that we achieved via a Chernoff bound. □

Proof of Lemma 2. An important subroutine is again the ability to achive a "tail" of size $\Theta(p)$ with a specified fill. The filler anchors $p-1$ cups, and then repeats the following algorithm for each anchored cup: At every step the filler adds one water to the anchored cups. For $p\log^2 p+1$ rounds the filler plays a single processor cup game on p cups with "negative" fill (i.e. fill below the average fill) allowed (it is strictly easier if negative fill is not allowed; we develop the more restrictive version because we will need to recursively apply the procedure). If among these games the emptier always neglects at least 1 anchor cup at least once, then the average fill of the anchor cups will have increased by $\Omega(\log^2 p)$, hence we have the desired backlog in an anchor cup. If on the other hand there exists a game where the emptier always empties a unit of water from each of the $p-1$ anchor cups, then the emptier and the filler are genuinely engaged in a single processor cup game on the remaining cups, in which by a well known construction the filler can achieve backlog at least $\log p$. Upon achieving such backlog the filler will swap this cup into the anchor set. At the conclusion of this process each anchor cup will have fill at least $\log p$ greater than the average fill of the non-anchor cups. However, this does not quite imply that the anchor cups have increased in fill by $\log p$ from the start of the process, as the average fill in the non-anchor cups has been depleted as water is siphoned off into the anchor cups. In particular, if the average fill of the non-anchor cups ends up as μ', started as μ, and the average fill has increased by $\epsilon \geq 0$ from μ ($\epsilon > 0$ could occur if cups zero out, which as remarked upon makes it strictly easier to achieve high fill), then we have $\mu' + (\log p)/2 \leq \mu + \epsilon$. If μ' is at least $(\log p)/2$ below the new average, then the average fill among anchor cups must be at least $(\log p)/2$ above this new average fill, which is at least the average fill. Hence by the end of this process all anchor cups are at least $(\log p)/2$ above the average fill at the beginning of the process.

After applying this process the filler cuts the number of processors in half and recurses on the anchor cups. Note that in recursion backlog is relative to the new average fill of the set that we recurse on. The recursion depth is $\log p$, so this strategy gives backlog at least $\frac{1}{2}(\log p + \log p/2 + \log p/2^2 + \cdots) \geq \Omega(\log^2 p)$ as desired.

□

Next we establish the Amplification Lemma:

Proof of Lemma 3. 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. □

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 $f_{k-1}(c/2), f_{k-1}(c/4), \ldots f_{k-1}(c/2^\alpha)$), 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 $1 \leq i \leq \alpha$, we have that $T(n) \leq \frac{\alpha n}{2} T(n/2)$. This recurrence yeilds $T(n) \leq poly(n)^{\log n} = O(2^{\log^2 n})$ 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 Lemma 1 can be improved to the following:

Lemma 4. *There exists an off-line filling strategy that achieves backlog $\Omega(2^{\sqrt{\log n}/2})$ with decent probability in time $O(n)$.*

Proof. TODO: do it, in particular, find XXX,YYY,ZZZ, and make this like legit We use Proposition 1. A lot. The basic idea is to basically use the Amplification Lemma for an oblivious filler. The only thing that changes is that we achieve the amplification a certain probability. In particular, it would be a bad idea to actually do the sum from the Amplification Lemma all the way down, becasue that kind of kills the probability that we get. Instead we opt to doing it like 9 times. So we have something like

If you can achieve backlog $f(k)$ with failure probability $g(k)$ then you can achieve backlog

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

with failure probability

$$g'(k) \leq g(k/2) + g(k/4) + \cdots + g(k/2^9)$$

where the probability bound is due to a union bound. Let's call this the ***oblivious amplification lemma***. Now we can proceed somewhat similarly as in the proof corollary of the amplification lemma in which we estabilised $\Omega(\text{poly}(n))$ backog was possible.

A major difference however is that we cannot continue the recursive construction of these f_m's all the way up to $f_{\log n^{1/9}}$, as this would require too much running time (superlinear). Also, we don't want the probability to decay too much, so we can't ever really go below $\sqrt{n}$ as the arg. Instead we recurse down from XXX all the way to ZZZ. Doing so we achieve backlog 2^{XXX}. With what probability? Well, the failure probability $g_{XXX}(k) \geq g_{XXX-1}(k/2)+g(k/4)+\cdots+g(k/2^9)$ Expanding the tree thing, and noting the upper limit on depth imposed by running time, and the lower limit on how much you can cut stuff imposed by the need for high probability, we get YYY leaves and stuff, and this is fine, because the failure probability was literally exponentially small in p so its hard to make it sad. Anyways, as desried, we have pretty decent backlog with pretty decent probabiliity.

OK, so wait a second. Let's say we use cutoff parameter α, and recursion depth β. Then the backlog is $(\alpha/2)^\beta$, and the running time is n^β. To get poly(n) run time we need $\beta \leq O(1)$, but to get $\Omega(2^{\sqrt{\log n}})$ backlog we neeed $\beta \geq \Omega(\sqrt{\log n})$. These are incompatible, so I don't think this is going to work out very well.

A much simpler solution: just lower n to $\sqrt{n}$ or whatever.

□