An investigation of variable-processor cup games

Alek Westover

February 23, 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 side 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: $O(\log\log n+\log p)$. 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, there exists a filling algorithm for achieving "amplified" backlog of*

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

We proceed to prove the Lemmas.

Proof of Lemma 1. First we establish a useful subroutine that the filler will use: acquiring a "tail" (a set of cups with fill at least some positive constant) of size $\Theta(p)$. The filler anchors $p-1$ cups and repeats the following algorithm for each anchored cup: At every step the filler adds one water to each anchored cup. For p^c rounds the filler plays a single processor cup game on 20 non-anchored cups that lasts for 19 steps. The filler's strategy will strongly disincentives the emptier from not emptying from each of the $p-1$ anchor cups every time, so for simplicity we initially will neglect the possibility that the emptier removes from multiple cups in the 20 cups. Over the 19 steps the emptier will remove from 19 cups. In the worst case these are all distinct members of the 20 cups. We can predict this sequence with constant probability. At the end of each of these single processor cupgames the filler with probability polynomially small in p swaps the cup that has received $\frac{1}{20}+\frac{1}{19}+\cdots+\frac{1}{1}>\ln 20$ fill with constant probability with a random cup in the anchor set. By repeating this polynomially many times the filler gets $\Theta(p)$ cups with fill at least ln20.

The filler follows the following algorithm:

1. Get $\Theta(p)$ cups with fill at least ln20
2. Using superpower to cut p to $p/2$ get $\Theta(p)$ *known* cups of fill at least $(\ln 20)/2$ (Note: this can be done *with high probability* by a chernoff bound. Let k be the probability that a singleprocessor cup game ends with a cup with fill at least ln20. Then the expected number of cups that have fill at least ln20 is $k\cdot p$, and by a chernoff bound it is exceedingly unlikely that we have less than say $k\cdot p/4$ cups with fill ln20. So the filler can safely assume that these $k\cdot p/4$ cups will occupy the emptier for long enough to get $k\cdot p/4$ known cups with fill at least $(\ln 20)/2$)

3. Recurse on those known cups

The maximum recursion depth possible is $\Theta(\log p)$ (we cut p in half each time we recurse), hence we have a backlog guarantee of $\Omega(\log p)$ as desired. □

Proof of Lemma 2. First we outline a construction to achieve backlog $\Omega(\log^2 p)$, then we generalize to achieve backlog $\Omega(p)$.

An important subroutine is again the ability to achive a "tail" of size $\Theta(p)$ with a specified fill. This is achieved in a similar manner as in the proof of Lemma 1. 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^{100} rounds the filler plays a single processor cup game on $1000p$ cups. If the emptier ever places at most 1 unit of water into the single processor cup game at each step then the filler swaps the cup which must have fill $\Omega(\log p)$ to the anchor set. If the emptier has placed multiple units of water in the single processor cup game in each of the games, then the anchor set has gained a ton of water. For now, for simplicity, we assume that we are playing against a greedy emptier so that this does not happen. Of course if the emptier is determined to not let the backlog in a single cup increase then it can, but at great cost. At the end of this process we have p cups of fill $\Omega(p)$. The filler then cuts p in half, and recurses. Recursion depth is of course $\log p$, so this strategy gives backlog $\Omega(\log^2 p)$.

□

Next we establish the Amplification Lemma:

Proof of Lemma 3. Anchor $p/2$ cups. By application of f, repeatedly get $p/2$ cups that have fill $f(p/2)$ and then swap these into the anchor set. Then cut p in half and recurse on the anchor cups. Note that the ability to increase p is needed in the recursive calling of f. Note that the $\frac{1}{2}$ factor happens because negative fill exists, i.e. the backlog outside of the anchor set is decreasig. But the anchor set is made relatively small so that this only has a small mulitplicative effect. □

We get the following as a corollary of the Amplication Lemma:

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 $\tilde{\Omega}(n^{1/9})$.

HMMMM, (4) to (5) seems kinda wasteful, we could keep more of the $\log_2$, also (1) to (2) is a bit wasteful.

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 $\Omega(n^{\frac{\log_2 3}{13}} \log_2 n)$.

□