[MMS] M. Manasse, L. McGeoch, D. Sleator. Competitive Algorithms for On-line Problems. *20th ACM Symp. on Theory of Computing*, 1988, 322–333.

[M] K. Mehlhorn. Data Structures and Algorithms, Vol.1. *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, 1984.

[RS] P. Raghavan and M. Snir. Memory vs. Randomization in Online Algorithms. *International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science No. 372, Springer-Verlag, 1989, 687–703.

[ST1] D. Sleator and R.E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *CACM* 28, Feb. 1985, 202-208.

[ST2] D. Sleator and R.E. Tarjan. Self-adjusting Binary Search Trees. *JACM* 32, 1985, 652-686.

[TV] R.E. Tarjan and C.J. Van Wyk. An $O(n \log\log n)$-time algorithm for triangulating a simple polygon. *SIAM Journal on Computing*, 17(1988), 143-178.

and substituting from equations 1 yields:

$$\overline{NewS_1OldS_1} \geq \overline{OldS_2FinalS_2} + 2\overline{NewA_2OldA_2}$$

which again is a weaker condition that the one imposed by Equation 3.

The above analysis has shown that

Theorem 4. *The above algorithm is 792-competitive.*

Proof: All the constraints in the analysis above can be satisfied by setting $a = 6, b = 7, c = 1, d = 14$, and $e = 28$. As noted earlier, when the adversary moves without forking, Φ increases by at most $\max\{22a, 8a + e\} = 22a$ times the distance moved. Thus, when the adversary moves without forking, the potential increases by at most 132 times the distance moved. Since algorithm $\mathcal{D}$ causes the adversary to do at most 6 times the work of an optimal adversary, the theorem follows. □

References

[BKT] P. Berman, H. Karloff, G. Tardos. A Competitive Three-Server Algorithm. *1st ACM-SIAM Symp. on Discrete Algorithms*, 1989, 280–290.

[BGRS] M. Bern, D. Greene, A. Raghunathan, M. Sudan. Online Algorithms for Locating Checkpoints. *22nd ACM Symp. on Theory of Computing*, 1990, 359–368.

[BLS] A. Borodin, N. Linial, M. Saks. An Optimal Online Algorithm for Metrical Task Systems. *19th ACM Symp. on Theory of Computing*, 1987, 373–382.

[CCF] A. Calderbank, E. Coffman, L. Flatto. Sequencing Problems in Two-server Systems. *Math. Oper. Research* 10, 1985, 585–598.

[CKPV] M. Chrobak, H. Karloff, T. Payne, S. Vishwanathan. New Results on Server Problems. *1st ACM-SIAM Symp. on Discrete Algorithms*, 1989, 291–300.

[CL] M. Chrobak and L. Larmore. An Optimal On-Line Algorithm for k Servers on Trees. manuscript, UC Riverside, 1989.

[C] R. Cole. On the Dynamic Finger Conjecture for Splay Trees. *22nd ACM Symp. on Theory of Computing*, 1990, 8–17.

[FRR] A. Fiat, Y. Rabani, Y. Ravid. Competitive k-Server Algorithms. 31st IEEE Symp. on Foundations of Computer Science, 1990, 454–463.

[HMRT] K. Hoffman, K. Mehlhorn, P. Rosenstiehl, R. Tarjan. Sorting Jordan Sequences in linear time using level linked search trees. *Information and Computation*, 68, 1986, 170-184.

[KMRS] A. Karlin, M. Manasse, L. Rudolph, D. Sleator. Competitive Snoopy Caching. *Algorithmica*, 3(1988), 79-119.

[L] J.M. Lucas. *Arbitrary splitting in splay trees.* Technical Report No. DCS-TR-234, Computer Science Department, Rutgers University, 1988.

for at the transition into State IV, these two distances are equal, S_1 being either at or to the right of A_1 by Lemma 12.

$$\overline{NewS_1NewA_2} \geq \overline{NewA_2FinalS_2} \tag{2}$$

for the state remains BALANCED at least until $FinalS_2$ is reached. To begin with, we suppose that $NewS_1$ and $NewS_2$ are coincident with or to the left of $OldS_1$ and $OldS_2$, respectively, and that $NewA_2$ is coincident with or to the right of $OldA_2$. Later, we show that the analysis of this case captures all the other cases.

Consider the motion of S_2 from $OldS_2$ to $NewS_2$. This reduced SAVED by at least $\overline{NewS_2OldS_2}$; it also increased $\overline{S_2S_3}$ by the same amount. Any motion of S_3 will also reduce SAVED, but it may reduce $\overline{S_2S_3}$ by up to an equal amount. So, in order to cause a player fork, S_1 must increase SAVED and hence BALSAVED by at least $2\overline{NewS_2OldS_2}$, if $NewS_2$ is to the left of $OldS_2$. But this increase to BALSAVED allows us to use the analysis of Case I to pay for the motion of S_2 from $NewS_2$ back to $OldS_2$. This argument shows that we can drop the assumption that $NewS_2$ is coincident with or to the left of $OldS_2$, for in either case it suffices to show how to pay for the motion of S_2 from $OldS_2$, or some point to its right, over to $FinalS_2$, assuming $FinalS_2$ is to the right of $OldS_2$. If $FinalS_2$ is to the left of $OldS_2$ the analysis of this paragraph has already shown how to pay for S_2 to reach $FinalS_2$.

We seek to use the same analysis as in Case III to pay for the motion of S_2 from $OldS_2$ to $FinalS_2$; to do this, we need Φ_{IV} to provide an additional term of size at least $2a\overline{OldS_2FinalS_2}$. But rewriting Equation 2 we have:

$$\overline{NewS_1OldS_1} + \overline{OldS_1OldA_2} + \overline{OldA_2NewA_2} \geq \overline{NewA_2OldS_2} + \overline{OldS_2FinalS_2}$$

and substituting from equations 1 yields:

$$\overline{NewS_1OldS_1} + 2\overline{OldA_2NewA_2} \geq \overline{OldS_2FinalS_2} \tag{3}$$

Now it is clear that the terms $d\overline{S_1OldS_1}$ and $e\overline{OldA_2A_2}$ at the start of the final move of Epoch E_3 provide the required term $2a\overline{OldS_2FinalS_2}$ if

$$d \geq 2a$$

and

$$e \geq 4a$$

It remains to consider the possibility that $NewS_1$ is to the right of $OldS_1$ or $NewA_2$ is to the left of $OldA_2$; if both are true, then $FinalS_2$ is to the left of $OldS_2$ and this case has already been analyzed. So suppose $NewS_1$ is to the right of $OldS_1$. Rewriting Equation 2 we have:

$$\overline{OldS_1OldA_2} - \overline{OldS_1NewS_1} + \overline{OldA_2NewA_2} \geq \overline{NewA_2OldS_2} + \overline{OldS_2FinalS_2}$$

and substituting from equations 1 yields:

$$2\overline{OldA_2NewA_2} \geq \overline{OldS_2FinalS_2} + \overline{OldS_1NewS_1}$$

which is a weaker condition that the one imposed by Equation 3.

Likewise, if $NewA_2$ is to the left of A_2, on rewriting Equation 2 we have:

$$\overline{NewS_1OldS_1} + \overline{OldS_1OldA_2} - \overline{NewA_2OldA_2} \geq \overline{NewA_2OldS_2} + \overline{OldS_2FinalS_2}$$

To facilitate the analysis of this case, we modify Algorithm $\mathcal{C}$ governing the adversary's moves to Algorithm $\mathcal{D}$. The new restriction is very simple: if the adversary does not perform a fork on the current move then only the server answering the request may move. It is readily seen that this does not affect the adversary's overall cost since any other adversary move can be deferred without increasing the total adversary costs.

Lemma 12. *S_1 satisfies one of the following conditions after serving the final request of epoch E_2.*

(i) S_1 coincides with A_1.

(ii) S_1 coincides with A_2.

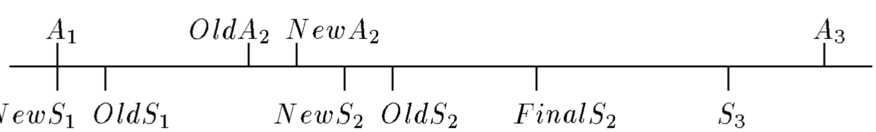
Proof. Suppose one of (i) or (ii) holds at some time during epoch E_2. Then, as the adversary is using algorithm $\mathcal{D}$, one of (i), (ii), or the following condition (iii) holds after serving each request of the epoch.

(iii) A_1 and A_2 are both to S_1's left (this may occur if there is a request at A_3).

If Move (a) or (b) applies, (ii) holds following the answering of the first request in epoch E_2. For if (a) applies, as algorithm $\mathcal{D}$ is being used, A_2 pseudo-forks to answer a request, which request is also answered by S_1. Again, if (b) applies, following the fork and any readjustment of position by A_1 or A_2 (see the definition of the adversary's algorithm $\mathcal{B}$), one of A_1 or A_2 and S_1 are still coincident. If Move (c) applies, $\text{SAVED} = 0$ at the start of the epoch. If there is no request served by S_1, SAVED remains equal to 0 and so Balanced State I is entered following the pseudo-fork. A request served by S_1 causes one of conditions (i)-(iii) to hold. Finally, if Move (d) applies, S_1 must answer some request, for otherwise there is not a separate epoch E_2 (see the definition of epochs), so again one of (i)-(iii) holds at some point.

If either (i) or (ii) holds at the end of epoch E_2, the proof is complete. So suppose (iii) holds. Consider the next move by A_2 (which causes the state transition); during this move, A_2 crosses S_1; but this can be viewed as the answering of a dummy request at S_1, and this request comprises the final request of epoch E_2; at this point (ii) holds. □

Recall that we are analyzing a player fork which causes a change of state from State IV. We note that this change of state could occur before the fork itself (for the motion of S_2 may bring A_2 closer to S_1 or A_1 than to S_2 itself). Basically, the problem is to provide additional terms proportional to $\overline{S_2S_3}$ in the potential Φ_{IV}. These additional terms are going to be supplied by the motion of S_1 and A_2. In order to describe this precisely, some additional notation is helpful. Define $OldS_1, OldA_2, OldS_2$ to be the positions occupied by S_1, A_2, S_2, respectively, when State IV is entered, i.e., when Epoch E_3 begins. Let $NewS_1, NewA_2, NewS_2$ denote the positions of S_1, A_2, S_2 at the start of the final move by S_2 in Epoch E_3, and let $FinalS_2$ denote the final position of S_2 in Epoch E_3.



The following relationships hold.

$$\overline{OldS_1OldA_2} = \overline{OldA_2OldS_2} \tag{1}$$

are to the left of A_1. In this event, we need to verify that the motion of S_2 to A_1 can be paid for. But then the term $2a\overline{S_2A_2} + a\overline{S_2S_1}$ decreases by a times the distance moved by S_2. In addition, S_3, the loner, moves towards A_1 by $\frac{1}{6}$ of the distance moved by S_2, increasing the term $4a \cdot \overline{S_3A_3}$ by $4a$ times the distance moved by S_3. So Φ decreases by at least the distance moved by S_2 if $a - \frac{4}{6}a \geq \frac{7}{6}$, that is if

$$2a \geq 7$$

Analysis for State III

When S_2 and S_3 meet, the potential at hand is reduced to the Case I potential and the analysis proceeds as in that case. It remains to analyze the motion prior to the fork.

There are two situations to consider. First, we consider the case where S_2 and S_3 are not sandwiched between A_2 and A_3; in other words, either A_2 is to the right of S_2 or A_3 is to the left of S_3. In this case, when S_2 and S_3 approach each other, the terms in Φ involving S_2 and S_3, including $16a \cdot \mathit{length}$ in *extra*, decrease by at least a times the distance moved. Here, by contrast with Case I, UNBALSAVED may decrease by up to the distance moved by S_2 and S_3. So Φ decreases by at least the distance moved if

$$a \geq c + 1$$

Next, we consider the case where S_2 and S_3 are sandwiched between A_2 and A_3; this can occur only in Case IIIa. In this case, both $\overline{S_2A_2}$ and $\overline{S_3A_3}$ increase. Thus, the terms involving the segments $\overline{S_2A_2}$, $\overline{S_3A_3}$ and $\overline{S_2S_3}$ increase Φ by a times the distance moved. In addition, if UNBALSAVED > 0, the resulting change to *extra* can increase Φ by up to c times the distance moved. Now, recall that Φ_{III} includes an extra $2a\overline{A_2A_3}$ term. We drop this term from Φ. Since $\overline{A_2A_3}$ is at least the distance moved (recall that S_2 and S_3 are sandwiched between A_2 and A_3), Φ decreases by at least the distance moved if

$$a \geq c + 1$$

Analysis for State IV

As in Case III, if one of S_2 or S_3 remains outside the interval $\overline{A_2A_3}$, the analysis of the fork does not need the extra terms in Φ_{IV}. The only difficult case is where S_2 and S_3 are sandwiched between A_2 and A_3. The extra terms are present to handle this case, specifically in order to compensate for the presence, in *extra*, of a non-zero UNBALSAVED term and, correspondingly, the absence of this component of SAVED in the BALSAVED term. The full details are quite involved.

Let E_3 be the State IV epoch with S_1 as the loner, let E_2 be the preceding epoch and let E_1 be the epoch preceding E_2. There are four moves by which epoch E_2 can be entered.

Move (a). E_1 was in a Balanced state (States I, III, or IV) with S_1 as the loner and A_2 pseudo-forks.

Move (b). E_1 was in a Balanced state with S_1 as the loner and A_2 forks.

Move (c). E_1 was in a Balanced state with S_3 as the loner and S_2 forks.

Move (d). E_1 was in a Balanced state with S_1 as the loner and S_2 moves to A_3 thereby making $\overline{A_2S_2} \geq \min\{\overline{S_1A_2}, \overline{A_1A_2}\}$.

are to the left of A_1. Then the term $2a\overline{S_2A_1} + a\overline{S_2S_1}$ decreases by a times the distance moved by S_2. In addition, S_3, the loner, moves towards A_1 by $\frac{1}{6}$ of the distance moved by S_2, increasing the term $4a \cdot l(A_3, A_2, S_3)$ by $4a$ times the distance moved by S_3. So Φ decreases by at least the distance moved by S_2 if $a - \frac{4}{6}a \geq \frac{7}{6}$, that is if

$$2a \geq 7$$

Analysis for Case Ib

- *Before the fork:* When S_2 and S_3 approach each other by the same distance, the following terms change in Φ_{Ib}: $10a \cdot \overline{A_3S_2}$ increases, while $2a\overline{A_3S_3}$, $a \cdot \overline{S_2S_3}$ and $16a \cdot length$ (part of *extra*) decrease, the latter since S_2 and S_3 are the two rightmost servers. *extra* also decreases due to the decrease in BALSAVED. Clearly, Φ decreases by $b + 5a$ times the combined distance moved by S_2 and S_3. So Φ decreases by at least the distance moved by S_2 and S_3 if
$$b + 5a \geq 1$$
- *After the fork:* The analysis is similar to that for Case Ia. Again, we switch to the potential Φ_{IIa}, even though State IIa may not be the current state. It is readily verified that the potential before the fork exceeds the potential following the fork by $8a\overline{A_3S_2}$. If the current state is not State IIa, then the remaining move by S_2 needs to be analyzed; but this is identical to the analysis in Case Ia and is not repeated here.

Analysis for Case Ic

The analysis is similar to that for Case Ib; there are two changes. First, before the fork, Φ decreases by $b + 3a$ times the combined distance moved by S_2 and S_3. So Φ decreases by at least the distance moved by S_2 and S_3 if

$$b + 3a \geq 1$$

Second, after the fork, Φ decreases by $4a\overline{A_2A_3} + 4a\overline{A_3S_2}$.

Analysis for Case II

- *Before the fork:* Recall that the potential Φ_{IIa} is being used, regardless of the actual state. Note that this does not cause a problem as there will not be a transition from State II prior to the fork. When S_2 and S_3 move towards each other the terms in Φ_{IIa} involving S_2 and S_3 decrease by at least a times the distance moved; UNBALSAVED decreases by at most the distance moved by S_2 and S_3. So Φ decreases by at least the distance moved by S_2 and S_3 if
$$a \geq c + 1$$
When S_2 and S_3 meet, the potential remaining is
$$4a \cdot l(A_1, A_2, S_1) + 4a\overline{S_3A_3} + extra$$
- *After the fork:* We use the potential Φ_{Ia} with S_3 as the loner. Since S_1 and S_2 are aligned, $4a \cdot l(A_1, A_2, S_1) = 2a\overline{A_2S_1} + 2a\overline{A_2A_1} + 2a\overline{A_1S_1} = 2a\overline{A_2S_2} + 2a\overline{A_2A_1} + 2a\overline{A_1S_1}$. Note also that SAVED $= 0$ and hence BALSAVED $=$ UNBALSAVED $= 0$. So the potential at hand is exactly what is needed. If the current state is not State Ia, then S_1 and S_2

- A_2 forks to A_1, the loner.
  The state and potential are unchanged.

Analysis for Cases III and IV

When the adversary forks in these two states, we simply drop the extra nonnegative terms that are added to the Case I potential to get the Case III and Case IV potentials, respectively. The potential is now that of Case I, and the analysis above applies. Clearly, the potential does not increase when the adversary forks in either of these two states.

6.2.3 Player Forks

Following the fork, the analysis for Cases I and II uses only potentials Φ_{Ia} and Φ_{IIa}, respectively. If a dummy request is the first request following the fork, then this restriction only applies until the dummy request. We note that when the request following the fork is served, the actual state will be either State Ia or IIa. In addition, if in State II before the fork, potential Φ_{IIa} will be used; but this can only reduce the actual potential.

It may be that as S_2 moves the state changes from State I to State II (there cannot be a transition from State II to State I for S_2 moves to the right, and hence increases $\overline{A_2S_2}$ if in State II). If there is a transition from State I to State II, the analysis is partitioned into two parts: one for the period in which State I applies and one for the period in which State II applies.

Analysis for Case Ia

- *Before the fork:* S_2 and S_3 meet to the left of A_3 (for otherwise State Ib must be entered prior to the fork; we can then consider the move to comprise a dummy request at A_3 followed by the actual request). The terms involving S_2 and S_3 increase Φ by at most a times the distance moved by S_2 and S_3. The term due to BALSAVED decreases Φ by b times this same distance. So Φ decreases by at least the distance moved by S_2 and S_3 if

$$b \geq a + 1$$

- *After the fork:* The request must have been answered by A_1 for if it was answered by A_2, S_2 would serve the request without forking. Immediately after the fork we want the potential Φ_{IIa} with S_3 as the loner:

$$4a \cdot l((A_3, A_2, S_3) + 2a\overline{S_2A_1} + 2a\overline{A_1S_1} + a\overline{S_2S_1} + \text{extra}$$

  Note that S_1 and S_2 are coincident. While immediately before the fork, the potential at hand was

$$4a\overline{S_1A_1} + 2a\overline{A_2S_2} + 2a\overline{A_2A_3} + 2a\overline{A_3S_3} + a\overline{S_2S_3} + \text{extra}$$

  and S_2 and S_3 were coincident. But these two potentials are equal. So following the fork a false State IIa may have been entered (for it need not be the case that A_1 is to the left of S_2 at this point); however, State IIa will be the actual state at the end of the move, since the player server that answers the request (be it S_1 or S_2) will be coincident with A_1. So it remains to consider how the remainder of the move affects the potential at hand assuming that the current state is not yet State IIa, which implies that S_1 and S_2

Analysis for Case Ib

- A_2 forks to A_1, the loner.
  Since we are in State Ib, only S_1 could have answered the request; thus, A_1 and S_1 are coincident. When A_2 forks to A_1, S_1, A_1 and A_2 are coincident; we shift to State IIb with S_1 as the player loner. It is easy to check that the Case Ib potential is larger than the Case IIb potential by $4a\overline{A_3S_2}$ at the fork.

- A_2 forks to A_3, the loner.
  This case does not arise as it first results in a transition to Case IIb.

Analysis for Case Ic

- A_2 forks to A_1, the loner.
  Since Case Ic applies, only S_1 could have answered the request; thus, A_1 and S_1 are coincident. When A_2 forks to A_1, S_1, A_1 and A_2 are coincident; State IIb is entered with S_1 as the player loner. Likewise, immediately prior to the fork, A_2 and A_3 are coincident. Now, it is easy to check that Φ_{Ic} and Φ_{IIb} are equal at the time of the fork.

- A_2 forks to A_3, the loner.
  This case does not arise for prior to the fork S_2 and A_3 would be coincident, which would have required a transition to Case Ib.

Analysis for Case IIa

- A_2 forks to A_3, the loner.
  The fork changes the state from IIa to IIIa, with S_1 as the loner. After the fork, $\overline{A_2A_3} = 0$, since A_2 and A_3 coincide. Before the fork, A_1 and A_2 coincide, thus the $4a \cdot l(A_1A_2S_1)$ term of Φ_{IIa} provides the $4\overline{S_1A_1}$ term of Φ_{IIIa}. Now it is easy to check that Φ_{IIa} immediately before the fork is equal to Φ_{IIIa} immediately after the fork.

- A_2 forks to A_1, the loner.
  This case does not occur as it would require a transition to Case Ia before the fork.

Analysis for Case IIb

- A_2 forks to A_3, the loner.
  S_2 and A_3 are coincident before the fork, causing a transition to Case IIa.

- A_2 forks to A_1, the loner.
  This case does not occur as A_2 and A_3 are coincident before the fork, causing a transition to Case Ib.

Analysis for Case IIc

- A_2 forks to A_3, the loner.
  The state and potential are unchanged.

- A_2 forks to A_1. In $\mathcal{B}$, A_2 moves to A_3 before forking to A_1. The actual cost is $\overline{A_2A_3}$, the change in Θ is $-\overline{A_2A_3}$, and the amortized cost is 0.

- A_2 forks to A_3. In $\mathcal{B}$, A_2 simply moves to A_3. Again, the amortized cost is 0, as in the above case.

- A_1 (resp. A_3) forks to A_2. Instead, A_1 (resp. A_3) moves to A_2. The actual cost is the distance moved ($\overline{A_1A_2}$ or $\overline{A_3A_2}$, respectively), and the potential decreases by at least the distance moved, since $\overline{A_1A_3}$ decreases by this amount. Thus, the amortized cost is nonpositive.

- Either A_1 forks to A_3 or vice versa. This can be analyzed in two parts. First, A_1 or A_3 moves to A_2, as in the above case. Then, A_2 forks to the corresponding loner, as covered in the first case. Clearly, the amortized cost is nonpositive.

We conclude that $\mathcal{B}$ does at most three times the work done by $\mathcal{A}$. □

Next, we introduce algorithm $\mathcal{C}$. In algorithm $\mathcal{C}$, as well as performing only the modified fork of algorithm $\mathcal{B}$, we require that prior to an adversary fork, the request is answered by both the player and the adversary loner, off which the fork is made, A_1 say; following the fork, A_1 may then return to its original position. Together with the modified fork, this causes the adversary to do at most twice the work done by Algorithm $\mathcal{B}$ and hence at most six times the work done by an optimal offline algorithm. In the analysis below, we assume that both the adversary loner and the player have answered the request; thus the state of the system is just prior to the (modified) fork.

Analysis for Case Ia

- A_1 is the loner when A_2 does a fork.
  Recall that either A_1 and S_1 or A_1 and S_2 or A_1 and S_3 coincide, immediately prior to the fork. If A_1 and S_1 coincide, the fork causes a shift from State Ia to State IIa. Φ_{Ia} and Φ_{IIa} are equal at the fork since at the instant that A_2 actually forks to A_1, A_2 and A_3 coincide (due to the modified fork). Thus, $\overline{S_2A_2} = \overline{S_2A_3}$ and $\Phi_{Ia} = \Phi_{IIa}$.
  If A_1 and S_2 or A_1 and S_3 coincide, the fork leaves the state unchanged. In the former case, the term $2a\overline{S_2A_2}$ immediately prior to the fork provides the term $2a\overline{A_2A_3}$ immediately after the fork (again, due to the modified fork). The other terms in the two potentials all match. In the latter case, the term $2a\overline{S_2A_2}$ is reduced by the fork by exactly the increase to the term $2a\overline{A_2A_3}$. Again, the other terms in the two potentials all match.
  So, in all three cases, the fork leaves the potential unchanged.

- A_2 forks to the loner A_3.
  Immediately prior to the fork A_1 and A_2 are coincident and are either to the right of or coincident with S_2 (for otherwise Case IIa applies). Consequently, the term $2a\overline{S_2A_2} + 2a\overline{A_2A_3}$ prior to the adversary's move is equal to the same term immediately following the fork; the other terms in the potential are unchanged. So the fork in this case leaves the potential unchanged.

Analysis for Case IIc

- The request is to the left of S_1.
  The analysis of this situation is identical to the analysis of the corresponding situation in State IIa.

- The request is between S_1 and S_2.
  If S_1 answers the request, the analysis is identical to that for a request to the left of S_1. If S_2 answers the request, then segment $\overline{A_3 S_2}$ decreases by the distance moved by S_2 and segment $\overline{S_2 S_3}$ increases by the distance moved by S_2. The term $l(A_1, A_2, A_3, S_1)$ does not increase despite S_1's movement, as S_1 must be to the left of A_3. In addition, UNBALSAVED decreases by at most $\frac{5}{6}$ of the distance moved by S_2. So Φ decreases by at least the distance moved by S_1 and S_2 if $a \geq \frac{7}{6} + \frac{5}{6}c$, that is if

$$6a \geq 7 + 5c$$

- The other two cases are impossible in State IIc, since no adversary server can be to the right of S_2.

Analysis for Case IV

By examining Φ_{IV}, we see that the analysis of this case is identical to that of State III, except in the situation where S_1 answers the request. In such a situation, the request must be at A_1 (otherwise, the current state is State II). The segment $\overline{S_1 A_1}$ decreases by the distance moved by S_1, while the segment $\overline{S_1 OldS_1}$ may increase by the distance moved by S_1. Furthermore, BALSAVED increases by the distance moved. So Φ decreases by at least the distance moved by S_1 if $4a - b - d \geq 1$, that is if

$$d \leq 4a - b - 1$$

6.2.2 Adversary Forks

For the purpose of this analysis, let A_2, the middle adversary server, be paired with the extreme adversary server that is closest to it, and let the other adversary server be the loner. We now show that an analogue of Lemma 3 holds, so again we only need use the following modified adversary fork—A_2 moves to the location of its paired server before forking to its loner. Forks by A_1 and A_3 are not used.

Lemma 11. *Given an algorithm $\mathcal{A}$, there is an algorithm $\mathcal{B}$ that employs the modified fork and performs at most three times as much work as $\mathcal{A}$ on the same request sequence.*

Proof: We assume that all three adversary servers are together in $\mathcal{A}$ at the start of the request sequence. We use a potential function Θ to facilitate this argument. In defining Θ, let A_3 be the server that is paired to A_2, and let A_1 be the loner. We set $\Theta = \overline{A_1 A_3} + \overline{A_2 A_3}$; Θ will be used to upper bound the work done by $\mathcal{B}$. The amortized cost of a move by $\mathcal{B}$ will be the sum of its actual cost plus the change in potential. Since Θ is 0 initially, and nonnegative at the end, the amortized cost of algorithm $\mathcal{B}$ is, indeed, an upper bound on its actual cost. When algorithm $\mathcal{A}$ moves without forking, algorithm $\mathcal{B}$ does the same. Their actual costs are the same, while the change in Θ might be twice the actual cost. Thus, the amortized cost of the move by algorithm $\mathcal{B}$ is at most three times the actual cost of $\mathcal{A}$. When $\mathcal{A}$ forks, its actual cost is 0. The actual cost of $\mathcal{B}$ on the same move needs careful analysis.

- The request is between S_1 and S_2.
Again, S_1 answers the request (otherwise, Case I applies). Again, $l(A_1, A_2, S_1)$ does not increase and UNBALSAVED increases by the distance moved by S_1. So Φ decreases by at least the distance moved by S_1 if
$$c \geq 1$$

- The request is between S_2 and S_3.
The request has to be at A_3, otherwise Case I applies. Recall that S_2 and S_3 move by the same distance towards the request. Thus, the segment $\overline{S_2A_3}$ decreases by the distance moved by S_2, the segment $\overline{A_3S_3}$ decreases by the distance moved by S_3, and the segment $\overline{S_2S_3}$ decreases by the combined distance moved. UNBALSAVED decreases by at most the combined distance moved. In summary, irrespective of whether S_2 or S_3 answers, Φ decreases by $3a$ times the combined distance moved and increases by c times the combined distance moved. So Φ decreases by at least the distance moved by S_2 and S_3 if
$$3a \geq c + 1$$

- The request is to the right of S_3; in this case it is at A_3.
The segment $\overline{S_3A_3}$ decreases by the distance moved by S_3, while the segment $\overline{S_2S_3}$ increases by the distance moved by S_3. Furthermore, UNBALSAVED decreases by at most the distance moved. So Φ decreases by at least the distance moved by S_3 if
$$a \geq c + 1$$

Analysis for Case IIb

- The request is to the left of S_1. The analysis of this situation is identical to the analysis of the corresponding situation in State IIa.

- The request is between S_1 and S_2.
If the request is at A_1 or A_2, S_1 answers the request (otherwise, Case I applies). Again, the analysis of this situation is identical to the analysis of the corresponding situation in State IIa.
If the request is at A_3, S_2 answers (otherwise Case IIc applies). Segment $\overline{A_3S_2}$ decreases by the distance moved by S_2, segment $\overline{S_2S_3}$ increased by the distance moved by S_2. Once again, S_1 moves $\frac{1}{6}$ of the distance moved by S_2; this may increase $l(A_1, A_2, S_1)$ by up to the distance moved by S_1. Also, UNBALSAVED decreases by at most $\frac{5}{6}$ of the distance moved by S_2. So Φ decreases by at least the distance moved by S_2 if $5a - \frac{4}{6}a \geq \frac{7}{6} + \frac{5}{6}c$, that is if
$$26a \geq 7 + 5c$$

- The request is between S_2 and S_3.
This is impossible in State IIb, since no adversary server can be between S_2 and S_3.

- The request is to the right of S_3.
Again, this situation is impossible.

most $\frac{4a}{6}$ times the distance moved by S_2. BALSAVED and UNBALSAVED behave as in the preceding case. So Φ decreases by at least the distance moved by S_1 and S_2 if $a - \frac{4a}{6} \geq \frac{7}{6} + \frac{5}{6}c$, that is if

$$2a \geq 7 + 5c$$

- The request is between S_2 and S_3.
  Whichever server answers, the terms involving S_2 and S_3 in Φ decrease by at least a times the distance moved. UNBALSAVED decreases by at most the distance moved by S_2 and S_3. So Φ decreases by at least the distance moved by S_2 and S_3 if

$$a \geq 1 + c$$

- The request is to the right of S_3. The argument of the item above, namely, when the request is between S_2 and S_3, applies.

Analysis for Case Ib

The same situations apply as in the analysis of Case Ia, with virtually the same analysis. The only new situations to consider are:

- The request is at A_3, between S_1 and S_2.

  (a) If S_1 answers, Case II applies.

  (b) If S_2 answers, the movement of S_2 to A_3 decreases segment $\overline{A_3S_2}$ and increases segment $\overline{S_2S_3}$. The movement of S_1 may increase segment $\overline{S_1A_1}$. Thus, Φ decreases by $9a$ times the distance moved by S_2 and increases by at most $4a$ times the distance moved by S_1. Again, BALSAVED can only decrease and UNBALSAVED decreases by at most $\frac{5}{6}$ of the distance moved by S_2. So Φ decreases by at least the distance moved by S_1 and S_2 if $9a - \frac{4a}{6} \geq \frac{7}{6} + \frac{5}{6}c$, that is if

$$50a \geq 7 + 5c$$

- The request is at A_1 or A_2 between S_1 and S_2.
  The player answers with S_2. By Lemma 4, when S_2 reaches A_3, State Ia is entered.

Analysis for Case Ic

The same situations apply as in the analysis of Case Ib, with virtually the same analysis. The only new situation to consider is with the request at A_3 (between S_1 and S_2) and S_2 answers. Then the movement of S_2 to A_3 decreases segment $\overline{A_3S_2}$ and increases segment $\overline{S_2S_3}$. The movement of S_1 increases segment $\overline{S_1A_1}$ by at most the distance moved by S_1. Again, UNBALSAVED decreases by at most $\frac{5}{6}$ of the distance moved by S_2. So Φ decreases by at least the distance moved by S_1 and S_2 if $5a - \frac{4a}{6} \geq \frac{7}{6} + \frac{5}{6}c$, that is if

$$26a \geq 7 + 5c$$

Analysis for Case IIa

- The request is to the left of S_1.
  The movement of S_1 does not increase $l(A_1, A_2, S_1)$; in addition, UNBALSAVED increases by the distance moved by S_1. So Φ decreases by at least the distance moved by S_1 if

$$c \geq 1$$

6.2.1 Player Moves Without Forking

It is possible for a player server to answer a request which causes it to cross an adversary server. To simplify the analysis, we consider this process to include dummy requests at the traversed adversary servers. In addition, as a request is served the state of the system may change. We analyze such a request in two (or more) phases: one phase for each of the states that hold as the request is serviced. Finally, we note the possibility of fractional moves in the player's strategy. The actual move always rounds down, but the player recalls the fractional moves that have not been made and as soon as a cummulative fraction reaches or exceeds unity the corresponding move is made. The analysis simply assumes fractional moves are made as they occur.

Analysis for Cases I and III

Each subcase that follows is taken to subsume the corresponding subcase of Case III as well as the named subcase of Case I.

Analysis for Case Ia

- The request is to the left of S_1 (so it must be at A_1).
The movement of S_1 decreases the segment $\overline{S_1A_1}$ by the distance moved, while BALSAVED increases by the distance moved. So Φ decreases by at least the distance moved by S_1 if

$$4a - b \geq 1$$

- The request is between S_1 and S_2.

(a) S_1 answers.
If the request is at A_1, the movement of S_1 decreases the segment $\overline{S_1A_1}$ and variable *length* by the distance moved, while BALSAVED increases by the distance moved. So Φ decreases by at least the distance moved by S_1 if

$$20a - b \geq 1$$

If the request is at A_2, then Case II would apply, since A_2 would be closer to S_1 than to S_2. Obviously, this also applies if the request is at A_3.

(b) S_2 answers.
If the request is at A_1, the movement of S_2 increases $2a\overline{S_2A_2} + a\overline{S_2S_3}$ by $3a$ times the distance moved by S_2 and the movement of S_1 decreases $4a\overline{S_1A_1} + 16a \cdot length$ by $\frac{20a}{6}$ times the distance moved by S_2 (recall that S_1 moves $\frac{1}{6}$ of the distance moved by S_2). While BALSAVED > 0, it decreases by $\frac{5}{6}$ of the distance moved by S_2 (the distance moved by S_2 minus the distance moved by S_1). If BALSAVED is 0, while UNBALSAVED > 0, it in turn decreases by $\frac{5}{6}$ of the distance moved by S_2. So Φ decreases by at least the distance moved by S_1 and S_2 if $-3a + \frac{20a}{6} \geq \frac{7}{6} + \frac{5}{6}c$, that is if

$$2a \geq 7 + 5c$$

If the request is at A_2, the movement of S_2 decreases $2a\overline{S_2A_2} + a\overline{S_2S_3}$ by a times the distance moved by S_2 and the movement of S_1 increases $4a\overline{S_1A_1} + 16a \cdot length$ by at forking so that the conditions of Case I apply. We refer to such a move as a *pseudo-fork*; clearly, this move cannot occur in the toy problem. Again, the same subcases apply as for Case I. The potential is given by

$$\Phi_{IV} = \Phi_I + d \cdot \overline{S_1 OldS_1} + e \cdot \overline{A_2 OldA_2}$$

$OldS_1$ and $OldA_2$ denote the locations of S_1 and A_2, respectively, at the moment State IV is entered. The function of the additional terms in the potential is to compensate for the $c \cdot$ UNBALSAVED term in the potential, as was the case for the $2a\overline{A_2A_3}$ term in Case III of the toy problem. This compensation is required only if the departure from State IV is via a player fork in which S_2 and S_3 are sandwiched between A_2 and A_3.

We refer to the state of the system as State I, II, III or IV when the corresponding case applies.

If, while in States III or IV, UNBALSAVED is reduced to zero, then State I is entered, and the additional terms in the potential are dropped.

The above definition needs to be modified in one instance. This matters in the analysis of Case IV in the event that a player forks. The modification is somewhat technical, and this paragraph can be omitted until the analysis of Case IV is read. The moves are partitioned into *epochs*. An epoch comprises the period in which a given state holds. We modify the above state definitions obtaining one case in which State II is not entered. Consider an entry to State IV, starting an epoch E_3 with S_1 as the loner; this entry must be by a pseudo-fork from the UNBALANCED state, State II, with S_1 as the loner. Consider the epoch, E_2, covering this UNBALANCED state. Let E_1 denote the epoch preceding E_2. Suppose that

(i) E_1 was in the BALANCED state with S_1 as the loner.

(ii) The transition from E_1 to E_2 resulted from a move of S_2 to A_3, which made $\min\{\overline{S_1A_2}, \overline{A_1A_2}\} \leq \overline{A_2S_2}$ (which entailed a transition from the BALANCED to the UNBALANCED state).

(iii) S_1 answered no requests during E_2.

Then we merge epochs E_1, E_2 and E_3, and we continue to use the potential for the original BALANCED state throughout. (That we can use this BALANCED state throughout follows from the fact that S_1 does not answer any request during E_2 and so has no effect on the analysis; to see this, one needs to consider the details of the analysis, which are given in Section 6.2.)

6.2 The Analysis

First, we note that each move by the adversary that does not involve forking or pseudo-forking increases Φ by at most a constant times the work done by the adversary. By examining Φ in each of the cases, this constant can be seen to be at most $\max\{22a, 8a + e\}$; later, we set $e = 4a$, making this constant at most $22a$. Likewise, a pseudo-fork by the adversary leaves Φ unchanged, since there is no change in Φ in transitions from one state to another, by Lemmas 5-10.

As before, in turn, we analyze (1) the scenario where an adversary server has answered the request and the player executes the algorithm, but does not fork, (2) the case where the adversary forks, and (3) the case where the player forks.

Lemma 10. *If* $\min\{\overline{A_1A_2}, \overline{S_1A_2}\}) = \overline{A_2S_2}$, $\Phi_{IIc} = \Phi_{Ic}$.

Proof: In Φ_{Ic}, $4a(\overline{A_2A_3} + \overline{A_3S_2}) = 4a\overline{A_2S_2} = 4a \cdot \min\{\overline{A_1A_2}, \overline{S_1A_2}\}$; together with the $4a\overline{S_1A_1}$ term and the remaining $4a\overline{A_2A_3}$ term from Φ_{Ic} this provides the $4a \cdot l(A_1, A_2, A_3, S_1)$ term in Φ_{IIc}. The equality of the remaining terms is readily checked. □

We comment briefly on our choice of potential function, which at first sight might appear rather arbitrary. Φ_{Ia} is the Case I potential for the toy problem with two changes. The first change is the addition of the term $2a\overline{A_2A_3}$; it is present to avoid negative terms in some of the other potentials. The second change is the addition of the term $16a \cdot length$; it is present to pay for the movement of S_1 towards S_2 when there is a request between them served by S_2 in the case that all three adversary servers are close to the player's pair. Again, Φ_{IIa} is the Case II potential for the toy problem with two changes: first, the introduction of the term $2a\overline{A_1A_2}$ (note that $4 \cdot l(A_1, A_2, S_1) = 2(\overline{A_1A_2} + 2\overline{S_1A_1} + \overline{S_1A_2})$); second, the introduction of the term $16a \cdot length$. These two terms need to be introduced to be consistent with Φ_{Ia} (to see this, consider a server fork from one case to the other). Given the choice of the partitioning into subcases (which may well not be the only choice) the remaining potentials are fixed, as we indicate below. To see this, it suffices to show that terms of the form $l(A_1, A_2, S_1)$ and $l(A_1, A_2, A_3, S_1)$ must be present in the potentials for Cases IIa and IIb and Case IIc, respectively. Then to see that the remaining potentials are forced it suffices to consider which terms can change in the transition from one state to another caused by the movement of A_2 and A_3.

We return to the question of the presence of terms $l(.)$. We consider the terms involving S_1 in Φ_{II}. Since the requests can force S_1 to go back and forth between several adversary servers it is clear the player can be made to do work while the adversary does none. So Φ needs to decrease. This is handled by a term such as UNBALSAVED. Suppose that S_1 were between A_1 and A_2 in Subcase IIa. If the terms in Φ_{IIa} involving S_1 increase as S_1 moves towards A_1 (resp. A_2), then they will decrease correspondingly as S_1 moves towards A_2 (resp. A_1); the increase has to be covered by the $c \cdot$ UNBALSAVED term in Φ_{IIa}. Clearly, the simplest solution is to have Φ_{IIa} neither increase nor decrease in this situation. The same argument applies in Case IIc, but to the interval A_1A_3. Suppose S_1 is between A_2 and A_3 in Case IIc. A movement of A_3 to the right eventually causes a transition to Case IIb. This implies that in Φ_{IIb}, if S_1 is to the right of A_2 the weight of interval $\overline{A_2S_1}$ should be equal to that of $\overline{A_1A_2}$. This yields a term proportional to $l(A_1, A_2, S_1)$ in Φ_{IIb}, with possibly an additional term proportional to $\overline{S_1A_1}$ if S_1 is to the left of A_1. The transition to Case IIa does not affect this term. By considering the transition from Case Ia to Case IIa via a server fork we see that we cannot afford an additional term proportional to $\overline{S_1A_1}$ when S_1 is to the left of A_1. This argument shows that the term $l(A_1, A_2, S_1)$ is the only term involving S_1 present in the potential for Cases IIa and IIb (apart from its implicit presence in the term *extra*). In Case IIc, the situation with S_1 to the right of A_3 does not have a transition to any other case, so it is natural to use the same weight for interval A_3S_1 as for interval A_1A_3, giving the term $l(A_1, A_2, A_3, S_1)$ for Case IIc.

Case III. Potential function for one type of special BALANCED.
This state occurs if the adversary forks from an UNBALANCED state. The same subcases apply as for Case I. The potential is given by

$$\Phi_{III} = \Phi_I + 2a\overline{A_2A_3}$$

Case IV. Potential function for the other type of special BALANCED.
This state occurs if the adversary or a server moves from an UNBALANCED state without

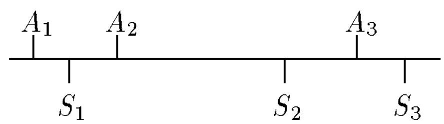
Lemma 5. *If* $\overline{A_2A_3} = \overline{A_3S_2}$, $\Phi_{Ib} = \Phi_{Ic}$.

Proof: Note that $4a\overline{A_2A_3} + 10a\overline{A_3S_2} = 8a\overline{A_2A_3} + 6a\overline{A_3S_2}$ when $\overline{A_2A_3} = \overline{A_3S_2}$. □

Case II. Potential function for UNBALANCED. $\overline{A_2S_2} \geq \min\{\overline{S_1A_2}, \overline{A_1A_2}\}$.

Let $l(X_1, X_2, \cdots, X_k)$ denote the length spanned by servers $X_1, X_2, \cdots, X_k$.

Subcase IIa. A_3 is to the right of S_2.

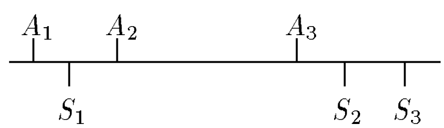


$$\Phi_{IIa} = 4a \cdot l(A_1, A_2, S_1) + 2a\overline{S_2A_3} + 2a\overline{A_3S_3} + a\overline{S_2S_3} + \text{extra}$$

Lemma 6. *If* $\overline{A_2S_2} = \min\{\overline{S_1A_2}, \overline{A_1A_2}\}$, $\Phi_{IIa} = \Phi_{Ia}$.

Proof: In Φ_{Ia} we write $2a\overline{A_2S_2}+2a\overline{A_2A_3}$ as $4a\overline{A_2S_2}+2a\overline{S_2A_3}$. But $4a\overline{A_2S_2} = 4a\min\{\overline{S_1A_2}, \overline{A_1A_2}\}$; together with the $4a\overline{S_1A_1}$ term from Φ_{Ia} this provides the $4a \cdot l(A_1, A_2, S_1)$ term in Φ_{IIa}. The equality of the remaining terms is readily checked. □

Subcase IIb. A_3 is to the left of S_2 and $\overline{A_3S_2} \leq \min\{\overline{S_1A_3}, \overline{A_2A_3}\}$.



$$\Phi_{IIb} = 4a \cdot l(A_1, A_2, S_1) + 6a\overline{A_3S_2} + 2a\overline{A_3S_3} + a\overline{S_2S_3} + \text{extra}$$

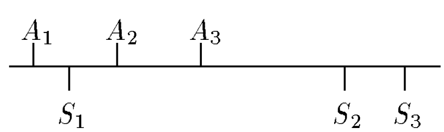
Lemma 7. *If* $\overline{A_2S_2} = \min\{\overline{S_1A_2}, \overline{A_1A_2}\}$, $\Phi_{IIb} = \Phi_{Ib}$.

Proof: In Φ_{Ib}, $4a(\overline{A_2A_3} + \overline{A_3S_2}) = 4a\overline{A_2S_2} = 4a\min\{\overline{A_1A_2}, \overline{S_1A_2}\}$; together with the $4a\overline{S_1A_1}$ term from Φ_{Ib} this provides the $4a \cdot l(A_1, A_2, S_1)$ term in Φ_{IIb}. The equality of the remaining terms is readily checked. □

Lemma 8. *If A_3 and S_2 coincide,* $\Phi_{IIa} = \Phi_{IIb}$.

Proof: If A_3 and S_2 coincide then $\overline{S_2A_3} = 0$; hence, $\Phi_{IIa} = \Phi_{IIb}$. □

Subcase IIc. A_3 is to the left of S_2 and $\overline{A_3S_2} \geq \min\{\overline{S_1A_3}, \overline{A_2A_3}\}$.



$$\Phi_{IIc} = 4a \cdot l(A_1, A_2, A_3, S_1) + 2a\overline{A_3S_2} + 2a\overline{A_3S_3} + a\overline{S_2S_3} + \text{extra}$$

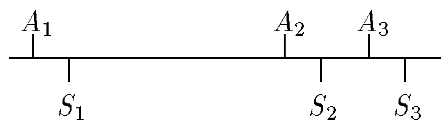
Lemma 9. *If* $\min\{\overline{A_2A_3}, \overline{S_1A_3}\} = \overline{A_3S_2}$, $\Phi_{IIb} = \Phi_{IIc}$.

Proof: The spare term $4a\overline{A_3S_2} = 4a \cdot \min\{\overline{A_2A_3}, \overline{S_1A_3}\}$ in Φ_{IIb} together with the term $4a \cdot l(A_1, A_2, S_1)$ equal the term $4a \cdot l(A_1, A_2, A_3, S_1)$ in Φ_{IIc}. Hence, $\Phi_{IIc} = \Phi_{IIb}$. □

extra = $16a \cdot \mathit{length} + b \cdot$ BALSAVED $- c \cdot$ UNBALSAVED. The definition of BALSAVED and UNBALSAVED are as before.

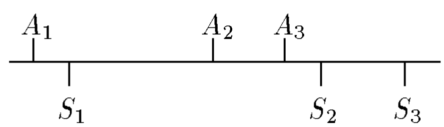
There are six major subcases. They are determined by the relative positions of A_2 and A_3. The BALANCED case occurs if A_2 is to the left of S_2 and closer to one of A_1 or S_1 than to S_2; otherwise the UNBALANCED case applies. Each of these cases is split into three subcases according as A_3 is to the right of S_2, to the left of S_2 but nearer S_2 than S_1 or A_2, and both to the left of S_2 and nearer one of S_1 or A_2 than S_2. The BALANCED case comes in several types according to whether it is entered by a player fork or otherwise; this is detailed further below.

Case I. Potential function for normal BALANCED. $\overline{A_2S_2} \leq \min\{\overline{S_1A_2}, \overline{A_1A_2}\}$ and SAVED = BALSAVED.
Subcase Ia. A_3 is to the right of S_2.



$$\Phi_{Ia} = 4a\overline{S_1A_1} + 2a\overline{A_2S_2} + 2a\overline{A_2A_3} + 2a\overline{A_3S_3} + a\overline{S_2S_3} + \text{extra}$$

Subcase Ib. A_3 is to the left of S_2 and $\overline{A_2A_3} \geq \overline{A_3S_2}$.

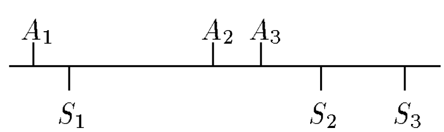


$$\Phi_{Ib} = 4a\overline{S_1A_1} + 4a\overline{A_2A_3} + 10a\overline{A_3S_2} + 2a\overline{A_3S_3} + a\overline{S_2S_3} + \text{extra}$$

Lemma 4. *If S_2 and A_3 coincide, $\Phi_{Ia} = \Phi_{Ib}$.*

Proof: When S_2 and A_3 coincide, $\overline{A_3S_2}$ is 0. Furthermore, $\overline{A_2S_2} = \overline{A_2A_3}$, implying that $2a\overline{A_2S_2} + 2a\overline{A_2A_3}$ in Φ_{Ia} equals $4a\overline{A_2A_3}$ in Φ_{Ib}. The equality of the remaining terms is readily checked. □

Subcase Ic. A_3 is to the left of S_2 and $\overline{A_2A_3} \leq \overline{A_3S_2}$.



$$\Phi_{Ic} = 4a\overline{S_1A_1} + 8a\overline{A_2A_3} + 6a\overline{A_3S_2} + 2a\overline{A_3S_3} + a\overline{S_2S_3} + \text{extra}$$

```
for each request r do
    (Case 1)
    if r is closest to one of the pair then
        Move the nearest server to r
        if r was between the loner and the middle server M then
            Move the loner towards r by 1/6 times the distance moved by M
        fi
        if r was between the pair then
            Move the other pair server towards r by the same distance
        fi
        Adjust SAVED accordingly
    fi
    (Case 2)
    if r is closest to the loner then
        if SAVED + distance(loner, r) is more than the pair distance then
            do
            Move loner towards r, increasing SAVED, until it equals the pair distance
            Move the pair to meet at their midpoint
            Fork one of pair to loner; (redefine loner and pair)
            Set SAVED = 0
            GoTo Case 1
            od
        else do
            Move loner to r
            Adjust SAVED accordingly
            od
        fi
    fi
```

Figure 2: Algorithm for the 3-server problem

6 A Competitive Algorithm for $k = 3$

In this section, we provide our main result: an algorithm for the original problem with $k = 3$. Recall that all three servers move on a line; a server may fork to a location occupied by another server at no cost. The basic intuition behind the algorithm is this: if one of the player's servers is "far" from the other two, we would like to act like they are on two different lines. In this case, an algorithm similar to the one provided for the toy problem would work well. So, can we somehow splice the two lines of the toy problem together, so all three servers move on the same line? The algorithm presented below, in Figure 2, does just this.

At any point in the algorithm below, the player has one designated loner server. The other two servers are, of course, called the pair. The loner is always one of the extreme servers (leftmost or rightmost). The middle server forks off the loner; immediately after the fork, the designated loner changes to the other end. SAVED, as before, is a nonnegative quantity, initially 0. When the loner moves SAVED increases by the distance moved by the loner. When one or both of the pair move, SAVED decreases by the distance moved by the pair, unless it is already 0. To start with, any one extreme server can be the first loner.

There is one aspect of the present algorithm not arising in the toy problem. In the situation in which the three adversary servers and the player pair are close together, but the loner is far away, it is possible to make the player do arbitrarily more work than the adversary unless the player's loner starts answering requests. To cause this, when the request is between the loner and the closer server of the pair (namely S_2), if the request is answered by S_2, then the loner moves a small proportionate distance towards S_2 (we have chosen to move S_1 by $\frac{1}{6}$ of the distance moved by S_2, though other fractions would work, with appropriate adjustments to the potential function). Recall that the player has no knowledge of the adversary and so must make this move regardless of the distribution of the adversary's servers. We remark that it is plausible that the same end could be achieved by a player fork of S_1 to S_2; this seemed more involved in its analysis, however, so we chose to avoid this additional use of a player fork.

6.1 The Potential Function

As in the toy problem there are two states: BALANCED and UNBALANCED. Loosely speaking, the state is UNBALANCED if at least two adversary servers are closer to the loner than to either of the pair. A precise definition of these states is given subsequently. The BALANCED state is partitioned into a normal BALANCED state and two special BALANCED states. One of the two special BALANCED states is essentially as in the toy problem; the other is similar in principle, although the details differ.

In order to start with an initial potential of zero, we require that all the servers, both the player's and adversary's, be aligned at the start. S_1 is chosen to be the loner.

We specify the potential function in full below. As before, the player's work reduces the potential by at least the work done, while the adversary's work increases the potential by at most the competitive factor times the amount of work done.

It is worth stressing that the description below assumes throughout that S_1 is the loner. Also, it should be kept in mind that both the adversary's and the player's servers are labeled so that the indices increase from left to right. Thus, for example, S_1 is always the leftmost player server, S_2 is always the middle player server and S_3 is always the rightmost player server.

We introduce the variable *length*: it is the separation between the two extreme servers (belonging to both the adversary and the player). In addition, we introduce the variable by up to this amount. So Φ decreases by at least the distance moved if

$$a \geq c + 1$$

We turn our attention to the case where the adversary forks. Under the rules of the toy problem presented earlier in this section, the adversary can fork one server of its pair to its loner at no cost. Let us define a *modified* adversary fork where the forking adversary server always moves to the location of its paired server before forking to its loner. Our analysis is further simplified by the following lemma, whose proof is similar to Lemma 1.

Lemma 3. *Given an algorithm $\mathcal{A}$ for the toy problem, there is an algorithm $\mathcal{B}$ that employs the modified fork and performs at most twice as much work as $\mathcal{A}$ on the same request sequence.*

In Case I, say that A_2 forks to A_1. By Lemma 3, we require that A_2 first move to A_3. The term $2a\overline{S_2A_2}$ in Φ_I provides the term $2a\overline{S_2A_3}$ in Φ_{II}. When A_2 forks to A_1, we divide the term $4a\overline{S_1A_1}$ in Φ_I equally to obtain the terms $2a\overline{S_1A_1}$ and $2a\overline{S_1A_2}$ in Φ_{II}. The other terms in Φ_I and Φ_{II} are identical. The same analysis handles the case of the adversary forking from Case III to Case II. When the adversary forks in Case II, we go to Case III. Again, assume that A_2 forks to A_3. The extra $2a\overline{A_2A_3}$ term is 0. The other terms are obtained by the reverse of the transformations used in the fork from Case I to Case II.

Now, we analyze the final case: the player forking. If the player forks in Case I, SAVED = BALSAVED = $\overline{S_2S_3}$. When S_2 and S_3 move to meet at their midpoint, Φ increases by at most a times the distance moved, since $2a\overline{S_2A_2}$ and $2a\overline{S_3A_3}$ together increase by at most $2a$ times the distance moved, while $a\overline{S_2S_3}$ decreases by a times the distance moved. Now, as SAVED is set to 0, Φ decreases by b times the distance moved. So Φ decreases by at least the distance moved by S_2 and S_3 if

$$b \geq a + 1$$

If the player forks in Case II, the movement of S_2 and S_3 to meet at their midpoint decreases Φ by at least a times the distance moved. Even if SAVED = UNBALSAVED, setting SAVED to 0 causes an increase in Φ by only c times the distance moved. So Φ decreases by at least the distance moved by S_2 and S_3 if

$$a \geq c + 1$$

Finally, if the player forks in Case III, we first drop the $2a\overline{A_2A_3}$ term. It is then easy to show that the movement of S_2 and S_3, together with this dropped term, causes a decrease in Φ by at least a times the distance moved. Again, even if SAVED = UNBALSAVED, setting SAVED to 0 causes an increase in Φ by only c times the distance moved. Once more, Φ decreases by at least the distance moved by S_2 and S_3, if

$$a \geq c + 1$$

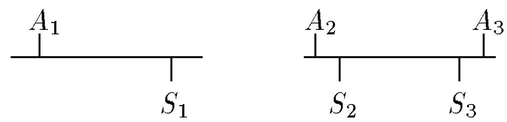
The reader can easily check that after S_2 and S_3 meet at their midpoints, the forking of S_2 to S_1 causes no change in Φ.

Theorem 3. *There is a 16-competitive algorithm for the toy problem of this section.*

Proof: All the constraints obtained in the above analysis can be satisfied by setting $a = 2$, $b = 3$ and $c = 1$. From Lemmas 2 and 3, the competitive factor is 16. □

Remark. In order to capture the cost of the initial potential in the bound of Theorem 3, we force it to be zero, which can be done as follows: we require the initial configuration to be in the BALANCED state, with servers S_1 and A_1 aligned, and with servers S_2, A_2, S_3, A_3 aligned.

Case III. Potential function for special BALANCED.



$$\Phi_{III} = \Phi_I + 2a\overline{A_2A_3}$$

First, we note the following easy fact.

Lemma 2. *When the adversary moves a server without forking, in each of the three cases* Φ *increases by at most* $4a$ *times the distance moved.*

We divide our analysis into three parts. First, we analyze the scenario where an adversary server has served the request and the player executes the algorithm, but does not fork. Then, in turn, we analyze the case where the adversary forks, and the case where the player forks.

To simplify the discussion, if a player server crosses an adversary server in answering a request, we create an intermediate imaginary request at the crossed adversary server.

We start by analyzing Cases I and III, with the player moving to answer requests. If S_1, the loner, moves to answer the request, then A_1 is already at the request site, having answered the request first. The movement of S_1 decreases the segment $\overline{S_1A_1}$ by the distance moved, while BALSAVED increases by the distance moved. So, Φ decreases by at least the distance moved by S_1, if

$$4a - b \geq 1$$

Next, we consider the situation in which one of the pair answers the request. First, suppose exactly one server of the pair moves in answering the request, S_2 say. $\overline{S_2S_3}$ increases by the distance moved and $\overline{S_2A_2}$ decreases by the distance moved. Further, while BALSAVED $>$ 0, it decreases by the distance moved. However, if BALSAVED $=$ 0, we start decreasing UNBALSAVED by the distance moved. So the net change to Φ is a decrease of at least $a - c$ times the distance moved. Second, suppose that both servers move in answering the request. Then the request is between the servers. $\overline{S_2S_3}$ decreases by the distance moved and $\overline{S_2A_2}+\overline{S_3A_3}$ is either unchanged or decreases, for one of the adversaries is at the request site. The change to BALSAVED and UNBALSAVED is as before. So again, the change to Φ is a decrease of at least $a-c$ times the distance moved. In either case, Φ decreases by at least the distance moved by S_2 and S_3, if

$$a - c \geq 1$$

Now, we analyze Case II, with the player moving to answer requests. If S_1, the loner, moves to answer the request, then either A_1 or A_2 (A_1, say) is already at the request site, having answered the request first. So, while $\overline{S_1A_1}$ decreases by the distance moved by S_1, $\overline{S_1A_2}$ increases by at most the same amount. In addition, UNBALSAVED increases by the distance moved, causing Φ to decrease by c times the distance moved. On adding the contributions to Φ, we see that if

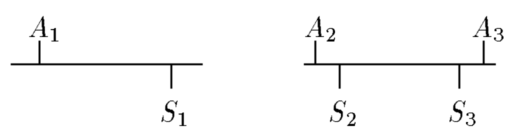
$$c \geq 1$$

then Φ decreases by at least the distance moved by S_1.

If either S_2 or S_3 (S_2, say) answers the request, $\overline{S_2S_3}$ increases by at most the distance moved, while $\overline{S_2A_3}$ decreases by the distance moved. Further, UNBALSAVED might decrease line occupied by the player's loner. If the adversary has two servers on the line occupied by the player's loner, the state is UNBALANCED. Note that the state can change in one of two ways — by the player forking or by the adversary forking. We further subclassify BALANCED into two substates: a "normal" BALANCED and a "special" BALANCED. Special BALANCED is the substate reached by the adversary forking into BALANCED from UNBALANCED. Otherwise, the state is normal BALANCED. We define three different potential functions, one for each state. The quantities BALSAVED and UNBALSAVED represent the amounts saved by the algorithm in the BALANCED and UNBALANCED states, respectively, since the last time the player forked. Thus, at any one time, SAVED is the sum of BALSAVED and UNBALSAVED. Note that the player and the algorithm are unaware of the division of SAVED into its two components. This division is only for the purposes of the analysis. For simplicity, we define the potential functions with S_1 as the loner; the case with S_3 as the loner is completely symmetric.

The need for the two terms BALSAVED and UNBALSAVED is shown by the following considerations. In the UNBALANCED state, by requests on the loner's line, the player can be forced to do work while the adversary does none; this can be paid for only by a decrease in the potential, a decrease which is proportional to the increase in SAVED. So in the UNBALANCED state, increases in SAVED need to be subtracted from the potential. Conversely, in the BALANCED state, a sufficient increase in SAVED will lead to a fork. The cost of bringing the pair together is not reflected in the adversary's move and so this cost must be extracted from the potential. As this cost is equal to SAVED, it suggests that the increases to SAVED need to be added to the potential when in the BALANCED state. Now we see that changes to SAVED are treated in opposite ways in the BALANCED and UNBALANCED states; hence the need for the two terms BALSAVED and UNBALSAVED in the potentials.

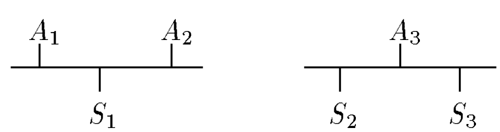
Case I. Potential function for normal BALANCED.



$$\Phi_I = 4a\overline{S_1A_1} + 2a\overline{S_2A_2} + 2a\overline{S_3A_3} + a\overline{S_2S_3} + b \cdot \text{BALSAVED} - c \cdot \text{UNBALSAVED}$$

Also, SAVED = BALSAVED.

Case II. Potential function for UNBALANCED.



$$\Phi_{II} = 2a\overline{S_1A_1} + 2a\overline{S_1A_2} + 2a\overline{S_2A_3} + 2a\overline{S_3A_3} + a\overline{S_2S_3} + b \cdot \text{BALSAVED} - c \cdot \text{UNBALSAVED}$$

```
for each request r do
    (Case 1)
    if r is on the line occupied by the pair then
        Move the nearest server to r
        if r was between the pair then
            Move the other pair server towards r by the same distance
        fi
        Adjust SAVED accordingly
    fi
    (Case 2)
    if r is on the line occupied by the loner then
        if SAVED + distance(loner, r) is more than the pair distance then
            do
            Move loner towards r, increasing SAVED, until it equals the pair distance
            Move the pair to meet at their midpoint
            Fork one of pair to loner; (redefine loner and pair)
            Set SAVED = 0
            GoTo Case 1
            od
        else do
            Move loner to r
            Adjust SAVED accordingly
            od
        fi
    fi
```

Figure 1: Program for the toy problem

essence of the difficulties introduced by forking, we introduce the following toy problem. Many of the algorithmic ideas and analysis techniques appearing in this section are used to solve the more general problem.

The toy problem we pose and solve can be abstracted as follows. We have *two* distinct lines, each of arbitrary length. We are presented with a request sequence $r_1, r_2, \ldots, r_n$. Each request is at some point on one of the two lines. Three mobile servers move on these lines to serve the request sequence. The servers obey the following rules. At any one time, we have to maintain at least one server on each of the two lines; thus, at any one time, two servers (called the *pair*) are located on one of the lines, while the third *lone* server, or *loner*, is located on the other line. A server may (1) move along one of the lines, incurring cost equal to the difference between its starting and ending positions; (2) fork, at no cost, to the position occupied by the loner, thus jumping lines and creating a new loner.

We emphasize that by the above forking rule, a server cannot fork to a location on the line on which it is presently located. Note also that the loner never forks.

We now present an online algorithm for this toy problem. The design of the algorithm is driven by the question of when to fork. Clearly, if the loner and two adversaries occupy the same line then the player can be forced to do arbitrarily more work, and so a fork is needed eventually. On the other hand the pair cannot be forced into doing excess work by the single adversary on their line. So an appropriate condition for determining when to fork would appear to be that the loner's work exceeds that of the pair by the cost of forking. The fork itself costs nothing; what we have in mind are the implicit costs of forking. To understand what these implicit costs might be let us consider the situation in which the loner shares its line with just one adversary. Then an "early" fork produces the previous undesirable situation. Indeed, it has led to an excess of work by the server equal to the previous separation of the pair. So this leads us to the following condition: the player will fork whenever the excess of work by the loner over the pair is equal to the separation of the pair. This excess is recorded in the variable SAVED.

We note one other point concerning the algorithm. We showed earlier, in Lemma 1, that if the adversary forks by bringing the forking server to a second server this at most doubles the cost of adversary's moves. The same is true for the server. So when forking, the server brings its pair together. As no one meeting location appears inherently more reasonable than any other, we chose to have the pair meet at their midpoint in a fork.

In the algorithm in Figure 1, SAVED is a nonnegative quantity that is initially 0. When the loner moves, it increases SAVED by the distance moved. When one (or both) of the pair moves, it decreases SAVED by the distance moved, unless SAVED is or goes to 0. In this case, SAVED remains at 0. Later, we show that this algorithm is 16-competitive.

In order to analyze the performance of this algorithm and later online algorithms, we think of player and adversary working in parallel on $r_1, r_2, \ldots, r_n$. For ease of analysis, the actions following the receipt of a request r ($= r_i$ for some i) are divided into the following steps: (1) the adversary moves arbitrarily; (2) the player executes an iteration of its algorithm.

We define a potential function Φ that is used to "smooth" the analysis. The player's work typically reduces Φ by at least the work done, while the adversary's work increases Φ by at most the competitive factor times the amount of work done. These bounds, along with bounds on the initial and final values of Φ, suffice to bound the player's work on sequence R_n by a multiple of the adversary's.

We define two *states* based on the relative positions of the player's servers and the adversary's servers. We say that the state is BALANCED if the adversary has only one server in the by $\mathcal{A}$ is the sum of the work done in each of the epochs. The work done by $\mathcal{A}$ in epoch E_i is at least the distance between its two servers at the end of epoch E_i. Let algorithm $\mathcal{B}$ process requests in the same manner as $\mathcal{A}$, except that at the end of each epoch the server that $\mathcal{A}$ would fork is moved by $\mathcal{B}$ to the location of the other server. By the argument above, $\mathcal{B}$ does at most twice as much work as $\mathcal{A}$, but does not employ the fork move. □

Theorem 1. *There is a 4-competitive algorithm for the case where $k = 2$.*

Proof: Chrobak et al. [CKPV] describe a 2-competitive algorithm for $k = 2$ against a non-forking adversary. By Lemma 1, this algorithm is 4-competitive against a forking adversary. □

4 A Lower Bound For a Non-Forking Player with $k \geq 3$

In this section we give a lower bound of $\Omega(m^{1/2})$ for an online algorithm that does not fork, for $k \geq 3$. Many of the ideas behind the proof below can be found in the lower bounds obtained by Bern et al. [BGRS].

Theorem 2. *For $k \geq 3$, any online algorithm that does not fork can be no better than $\Omega(m^{1/2})$-competitive.*

Proof: We give a proof for the case $k = 3$; the general case is similar but more involved.

The adversary incurs a one time cost of $m/2$ to position two of its servers distance $m/2$ apart.

Now we describe an adversary strategy that can be repeated in *epochs* of size $3\lceil m^{1/2}\rceil$. Each epoch will begin with two of the adversary servers at locations z_1 and $z_2 = z_1 + m/2$ and with history R_t. The adversary considers extending R_t by $3\lceil m^{1/2}\rceil$ alternating requests at z_1, z_2 and a third location which is one of $z_1 + m^{1/2}$ and $z_2 + m^{1/2}$. This third location is chosen so that the player has at most one server in the corresponding interval $[z_1 - m/4, z_1 + m/4]$ or $[z_2 - m/4, z_2 + m/4]$. Without loss of generality, let the third request location be $z_1 + m^{1/2}$. The adversary's solution is to leave two of its servers at z_1 and z_2 and to move its third server to $z_1 + m^{1/2}$, via a fork off the server at z_1, costing the adversary $m^{1/2}$; the adversary incurs no further costs in processing the epoch for when a server is present at a request site the cost of serving the request is zero. If the player does not fork, then it has two possible strategies. In the first strategy, one of its servers serves requests at both z_1 and $z_1 + m^{1/2}$, costing the player at least $(2\lceil m^{1/2}\rceil - 1)m^{1/2}$ for the epoch. In the second strategy, the player moves its servers so as to place one server at z_1 and another at $z_1 + m^{1/2}$, which costs at least $(m/4 - m^{1/2})$. In either case, the player is no better than $(\frac{m^{1/2}}{4} - 1)$ competitive.

After this epoch, the adversary repeats the pattern, possibly with the roles of $z_1 + m^{1/2}$ and $z_2 + m^{1/2}$ interchanged. □

5 A Toy Problem That Uses Forking

As mentioned earlier, the k-server framework introduced by Manasse, McGeoch and Sleator [MMS] does not allow forking. Although forking was introduced by Bern et al. [BGRS], none of the algorithms that they present and analyze use the fork move. The results of the previous section show us that for the problem under consideration, the player must fork. However, analyzing an algorithm that forks appears to be quite complicated. In order to capture the

The above results together establish the power of the fork move. In a scenario where forking is natural, any online algorithm for $k \geq 3$ fingers that does not utilize the fork move will compare poorly with an offline algorithm that forks.

Although the constant competitive algorithm presented for $k = 3$ is simple and easily implementable, its proof is quite complicated. In order to capture the essence of the difficulties introduced by forking, we abstract a "toy" problem and present an algorithm and its analysis in Section 5.

For $k > 3$, we do not yet have an online algorithm whose competitive factor is some function of k, although we conjecture that one exists. We feel that the techniques of this paper can be extended to obtain such an algorithm.

Apart from possible applications in finger search trees and splay trees, we believe our work introduces yet another interesting test case for competitive analysis. We are extending this type of analysis to a problem more difficult than those previously considered (caching, list access, online scheduling of elevators, disk drives etc.), as evidenced by our strong lower bound of $m^{1/2}$ for a non-forking online algorithm. Can competitive analysis nevertheless lead us to algorithms for finger searching that perform well in practice?

2 Preliminaries

In subsequent sections, we refer to the fingers as *servers*. In analyzing competitiveness, we usually refer to the online algorithm under consideration as the *player* and the optimal offline algorithm as the *adversary*.

Player servers and adversary servers (sometimes called simply *players* and *adversaries* when no confusion is possible) both move on a *line segment* (or simply, *line*). The line is undirected, so all servers can move in either direction on the line. For two points x and y on the line, $\overline{xy}$ is used to denote not only the line segment connecting them, but also its length.

The sequence of request locations is denoted $r_1, r_2, \ldots, r_n$, where each r_i is some location on the line. R_t denotes $r_1, \ldots, r_t$.

By the term *online*, we mean that the player chooses the positions of its servers at time t deterministically as a function of the request history R_t. The *offline* algorithm (i.e., the adversary) may use all of R_n to choose server locations at any time t. For an online algorithm P, let $Ratio(P, R_n)$ denote the ratio of the cost incurred by P to the cost incurred by the adversary on request sequence R_n. The *competitiveness* of online algorithm P is then defined to be the "worst ratio" $\sup_{n \to \infty} \sup_{R_n} Ratio(P, R_n)$.

3 A 4-Competitive Algorithm for k = 2

This section gives a 4-competitive algorithm for the case $k = 2$. One notable feature of this algorithm is that the player never forks, although the adversary may use the fork move.

Lemma 1. *Given an algorithm $\mathcal{A}$ that uses two servers and that employs the fork move to serve a request sequence $r_1, r_2, \ldots, r_n$, there is an algorithm $\mathcal{B}$ that does not fork and does at most twice the work of $\mathcal{A}$ to serve the same request sequence.*

Proof: We assume that the two servers of $\mathcal{A}$ are at the same location (*aligned*) at the start of the request sequence. Then we divide the request sequence into *epochs* $E_1, E_2, \ldots, E_n$. A new epoch is begun whenever $\mathcal{A}$ uses the fork move to process the next request. The work done we note that several conjectures concerning splay trees involve one or more fingers (eg. the Dynamic Finger Conjecture and the Split Conjecture [L]). Certainly, we are not claiming that the present investigation of fingers is directly connected to the splay tree conjectures, but it may shed light.

In this paper, we focus on the fork operation. To avoid additional complexities produced by a tree structure, we consider servers moving along a line. We model the problem as follows. We can afford k fingers. We are presented with a sequence of n requests, each at some location in a line segment of length m. A finger may (1) move along the line segment, incurring cost equal to the difference between starting and ending positions; (2) *fork*, at no cost, to a position currently occupied by another finger. Any number of these moves may be made in response to a request.

We analyze the *competitiveness* of our algorithms [BGRS, BLS, KMRS, MMS, ST]. That is, we compare the performance of an online algorithm against the performance of an optimal offline algorithm that sees all requests in advance. An algorithm is called c-competitive if its cost on any sequence of n requests is at most $O(1)$ greater than c times the offline algorithm's cost. This style of analysis refines traditional worst-case analysis. Competitive analysis is worst-case in that no assumptions about the distribution or correlation of requests are made; however, it measures performance relative to what is achievable by an omniscient algorithm, rather than in absolute terms.

A discretized version of our problem is an example of a *task system* as defined by Borodin et al. [BLS]. Borodin et al. give matching upper and lower bounds for a very general online problem, but their bound applied to our problem is no better than the trivial bound of m-competitiveness. Other related work includes a number of recent papers on *server problems* [BKT, BGRS, CCF, CKPV, CL, MMS, RS, FRR]. The two papers that address problems most similar to our work are by Bern et al. [BGRS] and by Chrobak et al. [CKPV]. The paper by Bern et al. [BGRS] presents optimally competitive algorithms for k servers that move on a line and can fork. However, their servers can move in only one direction. The paper by Chrobak et al. [CKPV] presents an optimally (i.e., k-) competitive algorithm for k servers moving on a line (this was generalized to k servers moving on a tree by Chrobak and Larmore [CL]). Our problem differs from this in that our servers can fork. The work of Bern et al. [BGRS] introduces the fork move to the server literature, although none of the algorithms that they present use this move. This move is very natural for applications in which servers represent information rather than physical objects.

We obtain the following results, comparing online algorithms against offline algorithms with an equal number of fingers:

- For the case $k = 2$, a 4-competitive algorithm. This algorithm has the interesting feature that it *never* employs the fork move even though the offline algorithm is allowed to fork (Section 3).
- For $k \geq 3$, a lower bound of $m^{1/2}$ for the competitive factor of any online algorithm that does not fork (Section 4).
- The main result of this paper, a constant competitive algorithm for $k = 3$ (Section 6). We emphasize that this constant factor is independent of m. By our current analysis, this constant factor is at most 792. We believe that it can be made much smaller than this.

Online Algorithms for Finger Searching

Richard Cole *   Arvind Raghunathan †

August 1, 1991

Abstract

A well-known technique for speeding up access into search structures is to maintain *fingers* that point to various locations of the search structure. The advantage of maintaining fingers is that accessing information near a finger is normally faster than the usual method of access. In this paper, we consider the problem of choosing locations in a large search structure at which to maintain fingers. In particular, we consider a server problem in which k servers move along a line segment of length m, where m is the number of keys in the search structure. Since fingers may be arbitrarily copied, we allow a server to jump, or *fork*, to a location currently occupied by another server. We present online algorithms and analyze their competitiveness. We show that the case $k = 2$ behaves differently from the case $k \geq 3$, by showing that there is a 4-competitive algorithm for $k = 2$ that never forks its fingers. For $k \geq 3$, we show that any online algorithm that does not fork its fingers can be at most $\Omega(m^{1/2})$-competitive. By contrast, in the main result of the paper, we show that for $k = 3$, there is an online algorithm that forks and is constant competitive (independent of m, the size of the search structure). Our algorithm is simple and implementable, although our analysis is complicated. The existence of a constant competitive algorithm for $k > 3$ is left as an open question.

1 Introduction

The use of fingers has proved particularly useful in balanced tree data structures; they have been applied in a diversity of areas, including for instance triangulation algorithms [TV] and Jordan sorting [HMRT]. Also, considerable attention has been given to designing finger search tree data structures (see [M]). We are interested in the following question: Given several fingers how best can they be used? In particular, we are interested in the fork operation: taking a finger from its present location to the location of a second finger.

A second, and more tenuous motivation for the investigation of fingers arises from the consideration of splay trees. Splay trees are a balanced tree scheme, devised by Sleator and Tarjan [ST2], which are efficient in an amortized sense. They conjectured that splay trees are as efficient as any binary search tree algorithm based on rotations. A special case of this conjecture, the Dynamic Finger Conjecture, was more or less completely resolved by Cole [C]. Unfortunately, Cole's proof was sufficiently complex, that we do not expect it to be extended to a proof of the general conjecture. So another approach may be helpful. In our opinion, understanding how to use fingers may provide such an approach. As supporting evidence

*Courant Institute, New York University, New York, NY 10012; this work was supported in part by NSF grants CCR-8902221 and CCR-8906949.

†Division of Computer Science, University of California, Davis, CA 95616