Data Location in the OceanStore

Sean C. Rhea **Wes Weimer**

January 13, 2000

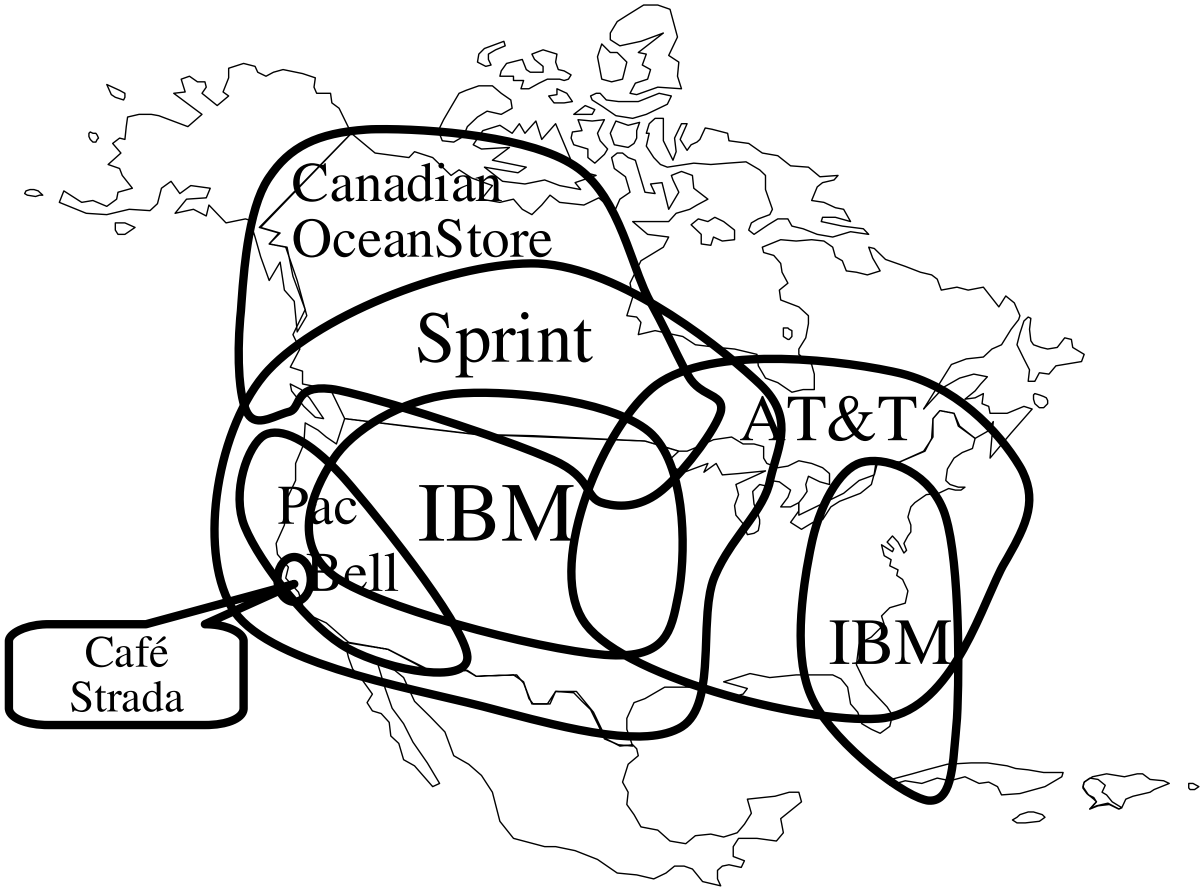
Overview

- OceanStore Primer
- The Data Location Problem
- Proposed Algorithm
- Analysis of the Proposed Algorithm
- Simulation Results
- Future Work
- Conclusion

OceanStore Primer

- Data storage can be provided like electricity.
- Although one specific provider is fiscally responsible for our data, it may be stored anywhere.
- Primary cost in the system is network communication.

OceanStore Primer (con't.)



OceanStore Primer (con't.)

Issues:

- All data must be encrypted. How can we handle conflict resolution on encrypted data?
- Data should "flow" to where it is used most often.

⇒ In the common case, we should be able to find data without referencing a central authority.

The Data Location Problem

Assumptions:

- All documents have some unique *virtual name*, which is independent of their physical location in the network.
- Their exists some deterministic, but expensive, algorithm for finding any document given this virtual name.
- Expenses within the OceanStore are measured primarily in terms of communication costs.

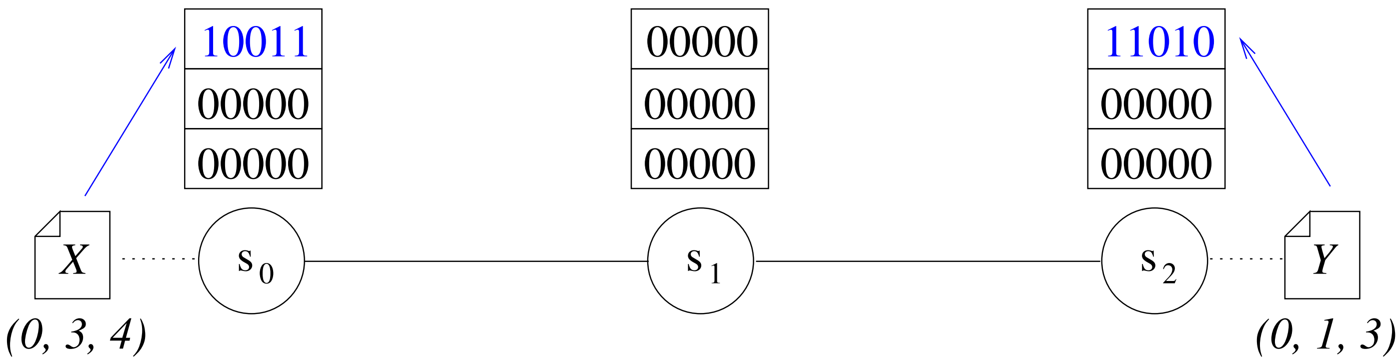
The Data Location Problem (con't.)

The idea:

- Use some inexpensive probabilistic algorithm to try and find the document. If it fails, fall back on the expensive but deterministic algorithm.
- Similar to the idea of using a cache to mitigate the cost of going to main memory.

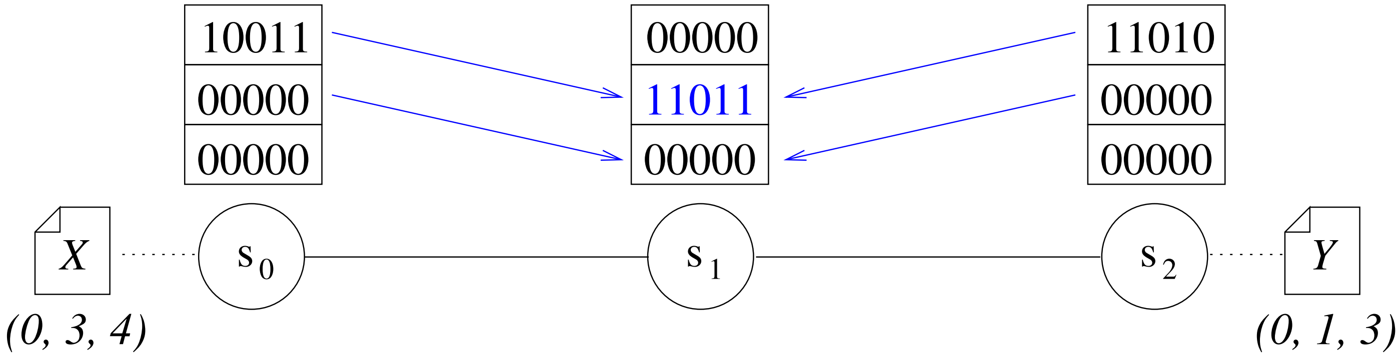
Proposed Algorithm — Update Phase (Step 1)

- Associate with each server a Bloom filter, which represents the set of documents on that server.
- Example:



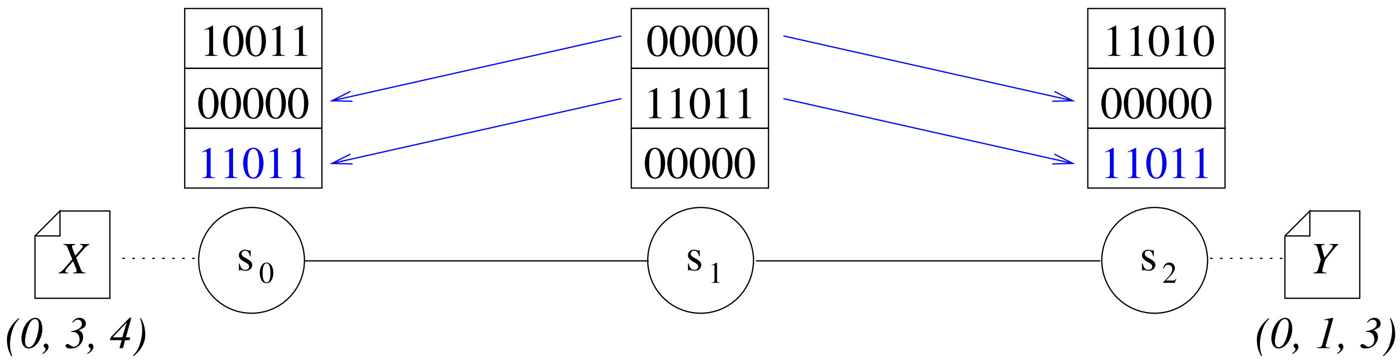
Proposed Algorithm — Update Phase (Step 2)

- Whenever the Bloom filter at a server changes, that server sends an *update message* containing the new value to all neighboring servers.
- In our example, the Bloom filters for s_0 and s_2 have both changed, so they send them to their collective neighbor, s_1, which combines them with the binary *OR* function in its second level.



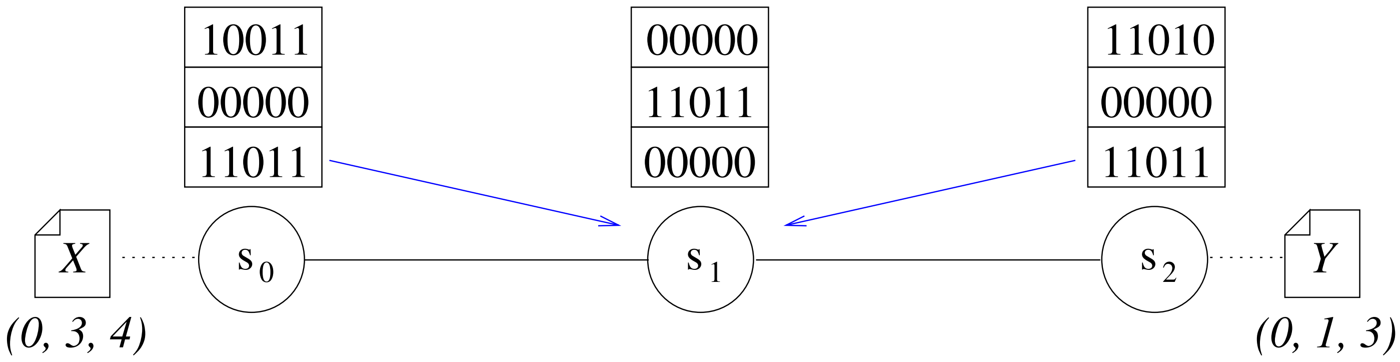
Proposed Algorithm — Update Phase (Step 3)

- The update messages continue until the Bloom filters at all servers stabilize.
- In our example, the Bloom filter for s_1 was changed last cycle, so it sends update messages to its neighbors this cycle.



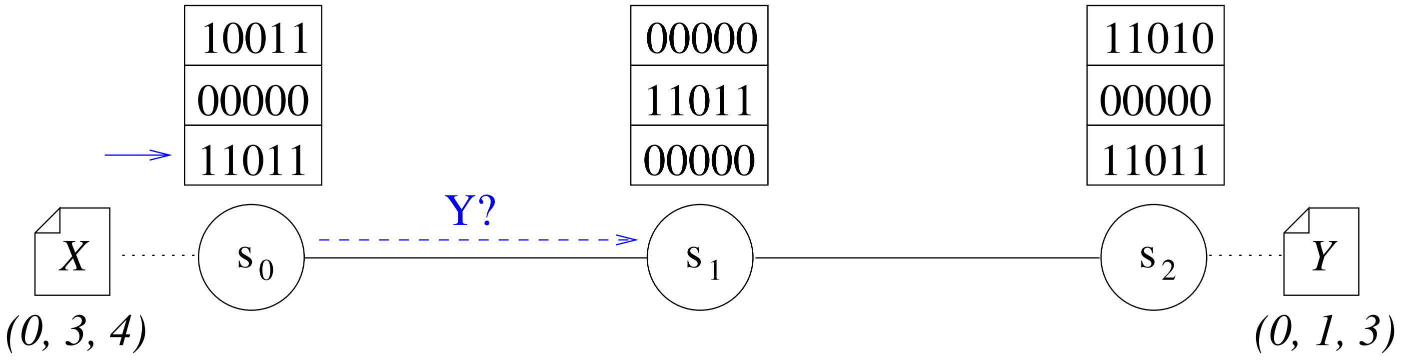
Proposed Algorithm — Update Phase (Step 4)

- Finally, although s_0 and s_2 have both changed again, they have not done so in a way that will effect s_1, so the update phase is complete.



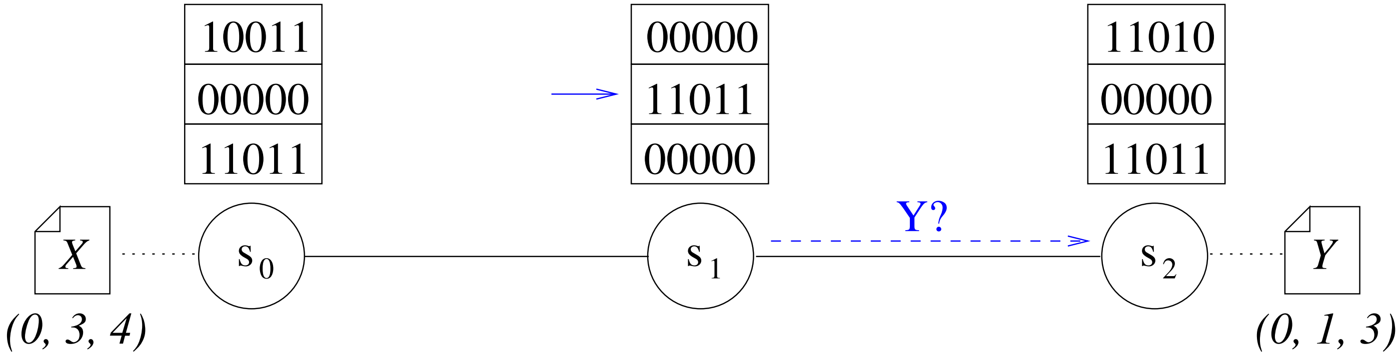
Proposed Algorithm — Query Phase (Step 1)

- A query is a directed depth–first search, guided by the Bloom filters.
- In this example, s_0 is trying to locate the document Y. Since the bits associated with Y are in s_o's Bloom filter at level 2, it is possible that Y is two steps away.
- s_0 sends a query message to its neighbor, s_1, to see if any of its neighbors have Y.



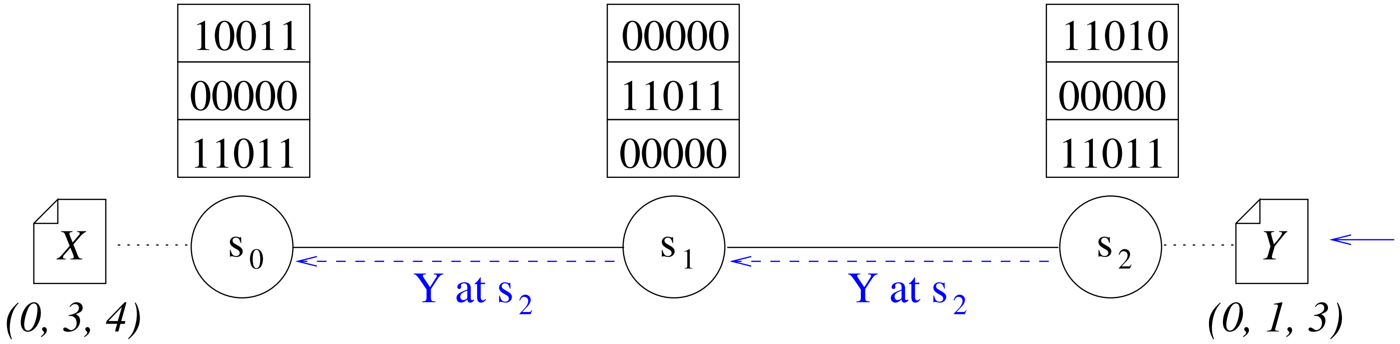
Proposed Algorithm — Query Phase (Step 2)

- s_1 sees from its Bloom filter that it definitely does *not* have Y, but that one of its neighbors one step away might.
- s_1 sends a query message to s_2, to see if it has Y.



Proposed Algorithm — Query Phase (Step 3)

- Since s_2 does indeed have Y, it can send a positive response back to s_0 through s_1.



Proposed Algorithm — Query Phase Remarks

- Each query message contains
  - the name of the document it is looking for
  - its point of origin
  - the current return path to that point
  - a list of all the nodes it has visited so far
- No node is ever visited twice, and the return path never grows larger than the Bloom filter depth.

Update Algorithm Convergence

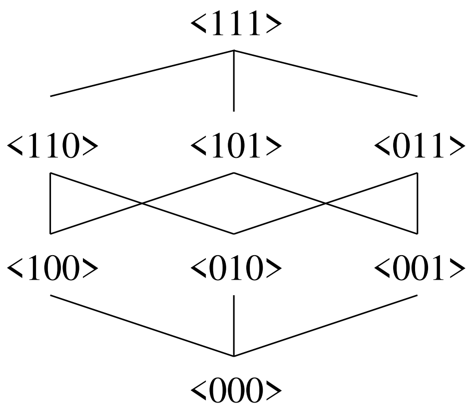
- Update algorithm (without document motion) settles in $\mathcal{O}(\mathit{width} \times \mathit{depth} \times \mathit{edges})$ even for cyclic graphs.

- Proof from compiler data-flow analysis:

  ◇ Attenuated Bloom filters (bit vectors) form a finite lattice;

  ◇ Update function is monotonic: always moves up lattice toward $< 111 >$;

  ◇ Therefore, only a finite number of updates can occur before all Bloom entries are ‘1’.



The lattice of
3-element bit vectors.

Update Algorithm Correctness

- Update function is a homomorphism on the lattice of Bloom filters. So updates can be reordered or delayed.
- The update algorithm simulates the forward data–flow analysis problem and thus computes the *join–over–all–paths* solution.
- Proof by extension of Kildall's 1973 data–flow work.
- So the Bloom filters contain the correct values when the algorithm settles.
- 'Correct' means no extra 1's and no incorrect 0's.

Update Algorithm with Motion

- In the presence of document motion, convergence is no longer guaranteed and Bloom filters may contain out–of–date information.

- This increases both the false positive rate and the false negative rate (which was previously zero!). Both increase the chance that our algorithm will fail to find a document that is actually within range.

Query Algorithm Correctness

- Query algorithm is greedy and can take suboptimal paths.
- Query algorithm can fail to reach the destination if there are too many false positives.
- This happens because we limit query path depth and never visit a node twice: if a query takes a long path (measured by edge–count) to a node when a shorter path exists, it cannot go back and visit the neighbors of that node via the shorter path later.
- Since our algorithm is greedy and takes the shortest path (measured by edge–count), this can only happen in the presence of false positives.

Simulation Setup

- Used the Boston University Web traces, which consist of entries like:

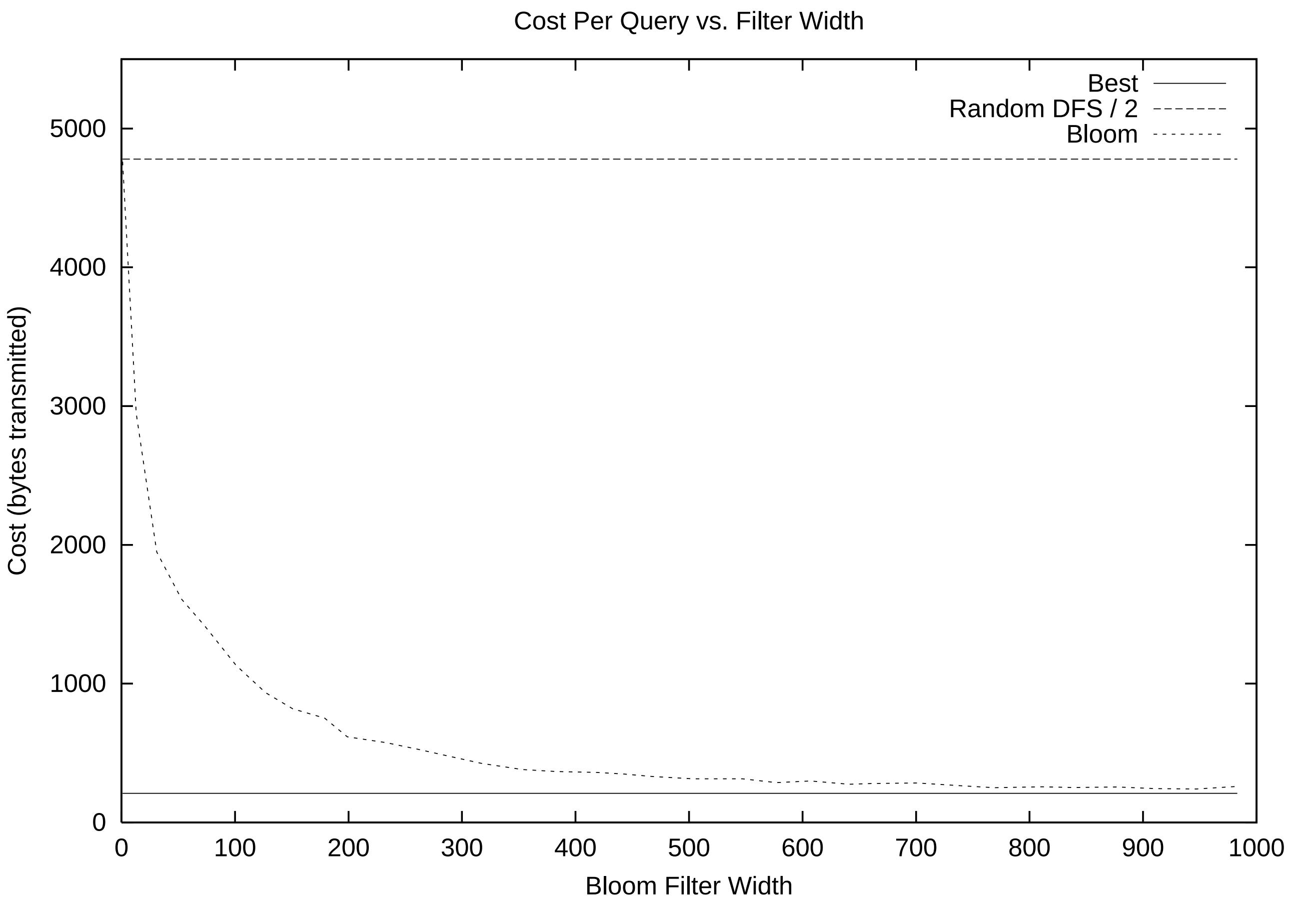
  `(time, client, user, URL)`

- For each different host in any URL, we added a server node. The server nodes were randomly connected between some minimum and maximum degree. Each client was connected to exactly one server.

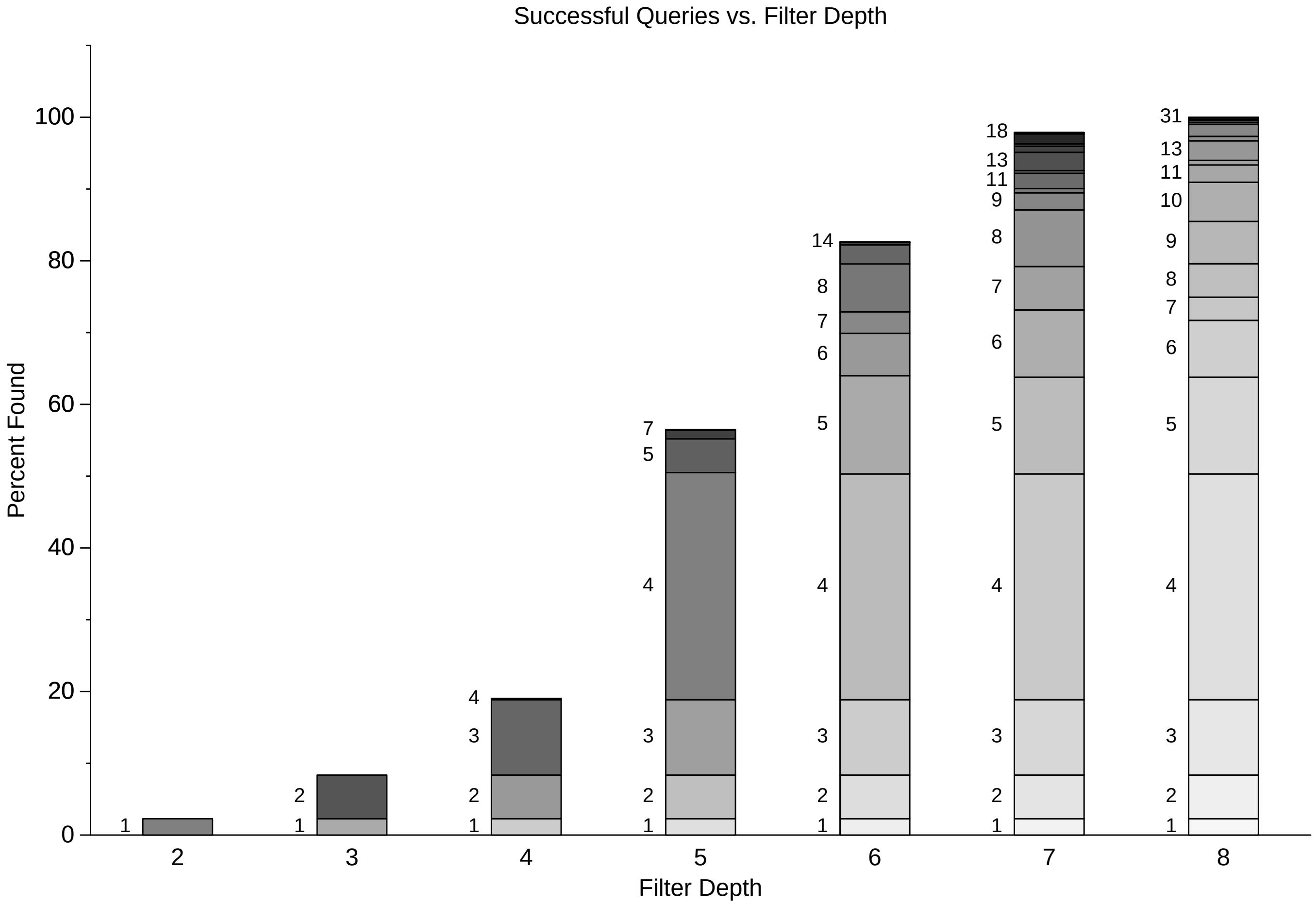
- The servers were allowed to update their Bloom filters until they settled.

- Finally, for each entry in the trace, the listed client performed a query for the listed URL.

Simulation Results — Cost vs. Filter Width



Simulation Results
Successful Queries vs. Filter Depth



Future Work

- Study interaction with the deterministic algorithm.
  - How long should we search before giving up?
  - What is the average path length of a successful query?
- Model the OceanStore notion of documents “flowing” to the places they are most used.
  - How does this effect the query algorithm?
  - Does the update algorithm still converge? How quickly?
- Simulate the algorithm in more detail.

Conclusion

- Attenuated Bloom filters are an effective method for directing localized searches.
- In almost all cases, they outperform a randomized depth–first search.
- For sufficient filter size, they approximate an oracle–directed search using only locally available information.