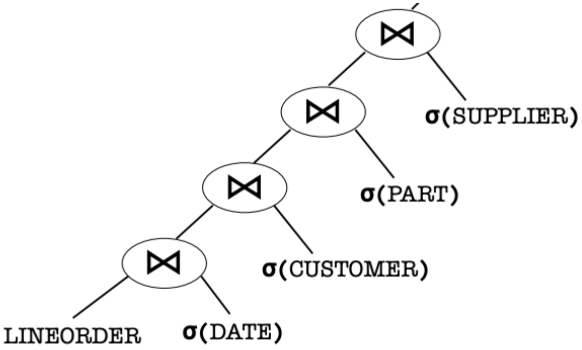
Accelerating Joins with Filters

Nicholas Corrado Xiating Ouyang

University of Wisconsin-Madison

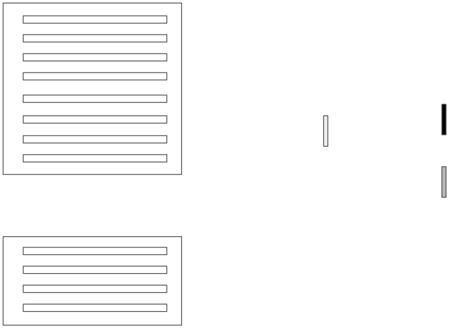
Star Schema



- If the query optimizer chooses a poor join order, intermediate join results may be unnecessarily large.
- Solution: try to filter out extraneous tuples before performing joins

Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)



Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)

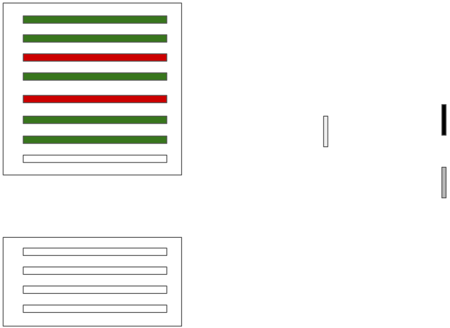
Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)

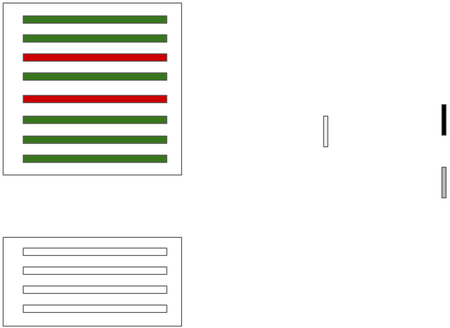
Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)



Lookahead Information Passing (LIP)

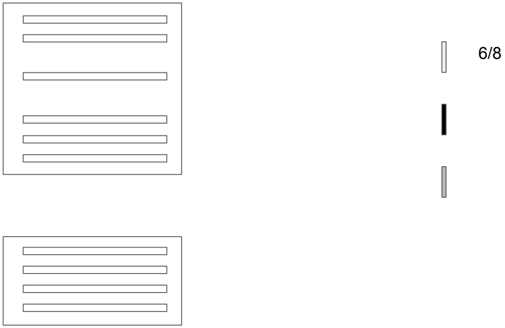


Lookahead Information Passing (LIP)

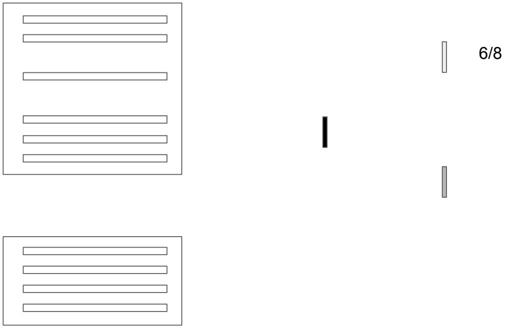
Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)

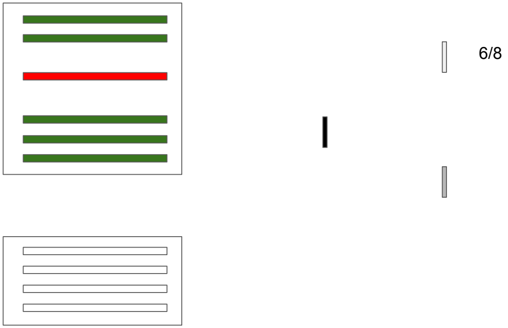
Lookahead Information Passing (LIP)



Lookahead Information Passing (LIP)



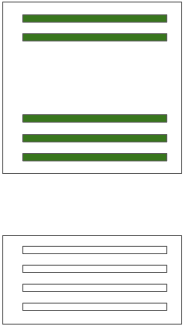
Lookahead Information Passing (LIP)



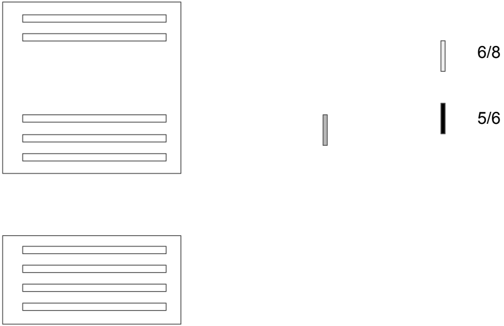
Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)



Lookahead Information Passing (LIP)



Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)

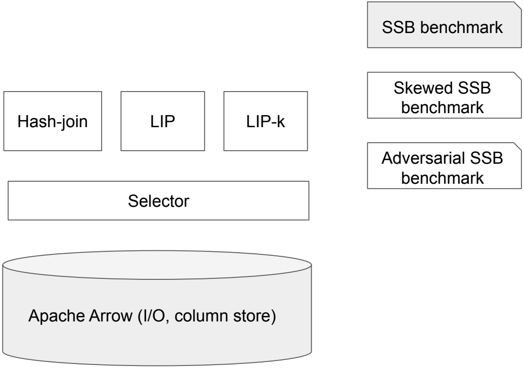
Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)

Lookahead Information Passing (LIP)

- LIP uses statistics from all previous batches to compute σ
  - Slow response to local changes in key distributions
- **LIP-k**: Only use the previous k batches to compute σ

Implementation and benchmarking



An Example Experiment

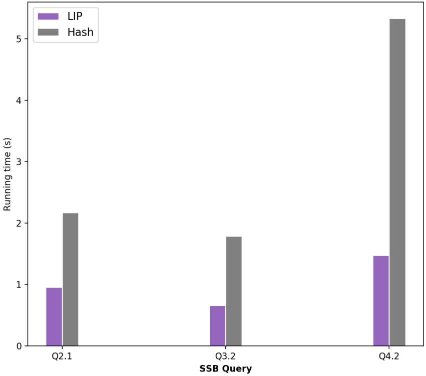
- Select where `Credit Score` ≥ 700

`Credit Score`	
720	
750	50 batches
720	
...	
400	
400	50 batches
400	
...	
770	
810	50 batches
800	
...	

An Example Experiment

- Select where `Credit Score` ≥ 700

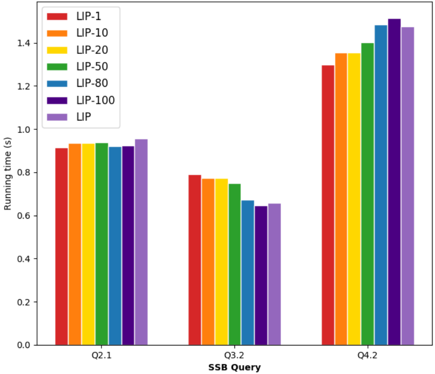
`Credit Score`	
720 750 720 ...	50 batches
400 400 400 ...	50 batches
770 810 800 ...	50 batches



An Example Experiment

- Select where `Credit Score` ≥ 700

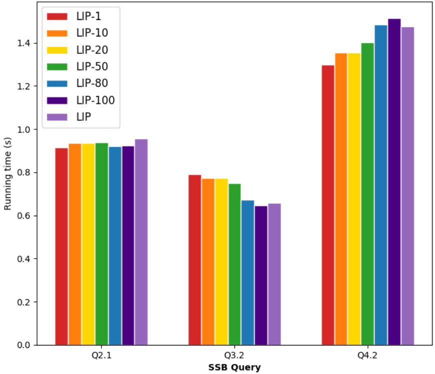
`Credit Score`	
720	50 batches
750	
720	
...	
400	50 batches
400	
400	
...	
770	50 batches
810	
800	
...	



An Example Experiment

- Select where `Credit Score` ≥ 700

`Credit Score`	
720	
750	
720	50 batches
...	
400	
400	
400	50 batches
...	
770	
810	
800	50 batches
...	

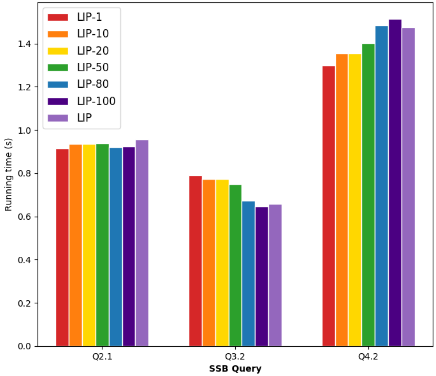


- LIP-k performs better than LIP on some queries...

An Example Experiment

- Select where `Credit Score` ≥ 700

`Credit Score`	
720 750 720 ...	50 batches
400 400 400 ...	50 batches
770 810 800 ...	50 batches



- LIP-k performs better than LIP on some queries...
- ...but LIP performs better on others

LIP is solving an online problem

- Given any tuple t, a mechanism $\mathcal{M}$ decides a sequence of applying the filters to *minimize* the number of probes.

LIP is solving an online problem

- Given any tuple t, a mechanism $\mathcal{M}$ decides a sequence of applying the filters to *minimize* the number of probes.
  - if t passes **all** filters: n probes necessary

LIP is solving an online problem

- Given any tuple t, a mechanism $\mathcal{M}$ decides a sequence of applying the filters to *minimize* the number of probes.
  - if t passes **all** filters: n probes necessary
  - if not, at least one filter rejects it: 1 probe best / n probes worst

LIP is solving an online problem

- Given any tuple t, a mechanism $\mathcal{M}$ decides a sequence of applying the filters to *minimize* the number of probes.
  - if t passes **all** filters: n probes necessary
  - if not, at least one filter rejects it: 1 probe best / n probes worst

$$\frac{\#\text{probes by } \mathcal{M}}{\#\text{probes by OPT}} \leq n.$$

LIP is solving an online problem

- Given any tuple t, a mechanism $\mathcal{M}$ decides a sequence of applying the filters to *minimize* the number of probes.
  - if t passes **all** filters: n probes necessary
  - if not, at least one filter rejects it: 1 probe best / n probes worst

$$\texttt{Competitive ratio of } \mathcal{M} = \frac{\#\text{probes by } \mathcal{M}}{\#\text{probes by OPT}} \leq n.$$

LIP is solving an online problem

- Given any tuple t, a mechanism $\mathcal{M}$ decides a sequence of applying the filters to *minimize* the number of probes.
    - if t passes **all** filters: n probes necessary
    - if not, at least one filter rejects it: 1 probe best / n probes worst

$$\texttt{Competitive ratio of } \mathcal{M} = \frac{\#\text{probes by } \mathcal{M}}{\#\text{probes by OPT}} \leq n.$$

Theorem

There is no **deterministic** *mechanism $\mathcal{M}$ for LIP achieving a competitive ratio less than N, where N is the number of filters used in LIP.*

LIP is solving an online problem

- Given any tuple t, a mechanism $\mathcal{M}$ decides a sequence of applying the filters to *minimize* the number of probes.
  - if t passes **all** filters: n probes necessary
  - if not, at least one filter rejects it: 1 probe best / n probes worst

$$\texttt{Competitive ratio of } \mathcal{M} = \frac{\#\text{probes by } \mathcal{M}}{\#\text{probes by OPT}} \leq n.$$

Theorem

There is no **deterministic** *mechanism $\mathcal{M}$ for LIP achieving a competitive ratio less than N, where N is the number of filters used in LIP.*

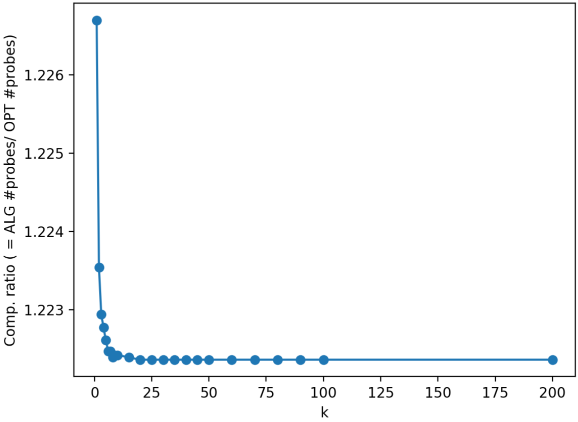
- Randomness?

Conclusion

- Implemented LIP and its variant LIP-k
- Relative performance of LIP and LIP-k depends on the query
- Can we use randomness to achieve a better robustness guarantee?

Thank you!

Competitive Ratio vs. k on Uniform Data



Competitive Ratio vs. k on Adversarial Data

- Adversarial data set constructed such that LIP-k has worst case performance for odd k
- Run on query with $N = 2$ joins