Poor Man's Social Network

Consistently Trade Freshness For Scalability

Zhiwu Xie, Jinyang Liu, Herbert Van de Sompel, Johann van Reenen and Ramiro Jordan
Outline

- Scaling feed following
- Algorithm
- Experiment and results
- Conclusions
Feed Following Scalability

Give me the 20 most recent tweets sent by all the people I follow

- Individualized queries
- Fast changing global state
- Partitioning, replication, and caching
- NoSQL: trade consistency for scalability
Consistency

- Atomicity, Linearizability, or One-copy Serializability (1SR)
Retweet Anomaly

Feed Following:

blah

Retweet: blah

Feed Following:

Retweet: blah

blah
New Approach: TimeMap Query

**Who** have created new tweets during the **past scheduled release** periods?

- Global time across partitions
- Schedule releasing
- Client-side processing and caching
- Consistently trade freshness for scalability
**CAP Theorem**

- Preconditioned on the asynchronous network model: the only way to coordinate the distributed nodes is to pass messages.
- In the partially synchronous model, where global time is assumed to be available, CAP may indeed be simultaneously achievable most of the time.
Global Time

• “One of the mysteries of the universe is that it is possible to construct a system of physical clocks which, running quite independently of one another, will satisfy the Strong Clock Condition.”

– Time, Clocks and the Ordering of Events in a Distributed System, by Leslie Lamport
Who have created new tweets during the past scheduled release periods?
Partitioning: Send A New Tweet

User_id: 0, 5, 10, 15,…
User_id: 1, 6, 11, 16,…
User_id: 2, 7, 12, 17,…
User_id: 3, 8, 13, 18,…
User_id: 4, 9, 14, 19,…
Partitioning: TimeMap
If the current time is 1:05:37PM, please tell me who (no matter if I follow any of them or not) have sent new tweets from 1:05:30PM to 1:05:35PM. I'll figure out by myself if any of these new tweets are relevant to me, and if so, I'll retrieve these tweets separately by myself.

If the current time is 1:05:39PM, please tell me who (no matter if I follow any of them or not) have sent new tweets from 1:05:30PM to 1:05:35PM. I'll figure out by myself if any of these new tweets are relevant to me, and if so, I'll retrieve these tweets separately by myself.

Cache!
Staleness vs. Latency

How are you?
I'm fine (as of 2:00)

How were you at 12:55?
I was fine (as of 12:55)

1:00 2:00

1:00 1:05

Fresh, but 1 hour latency

10 minutes stale but only 5 minutes latency
Trade Freshness For Scalability

- Mass transit system vs. private car
- Lose flexibility, but gain overall efficiency by sharing resources
- Stale up to the length of the schedule release period, e.g., 5 seconds.
Experiment

- Implemented on AWS
- A Twitter like feed following application
- Server side: Python/Django, PostgreSQL, PL/pgSQL
- Client side: emulated browser, implemented in Python/Django and PostgreSQL
Experiment: Configurations

- Used ~ 100 cloud instances from Amazon
- Most are used for emulated browsers
- 3 to 6 c1.medium as servers
- Use memcached to simulate caches
Experiment: Workload

- Work load similar to the Yahoo! PNUTS experiment
- A following network of ~ 200,000 users
- Synthetic workload generated by Yahoo! Cloud Serving Benchmark

<table>
<thead>
<tr>
<th></th>
<th>PNUTS</th>
<th>This</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of producers</td>
<td>67,921</td>
<td>67,882</td>
</tr>
<tr>
<td>Number of consumers</td>
<td>200,000</td>
<td>196,283</td>
</tr>
<tr>
<td>Consumers per producer</td>
<td>15.0</td>
<td>13.38</td>
</tr>
<tr>
<td>Zipf parameter</td>
<td>0.39</td>
<td>0.39</td>
</tr>
<tr>
<td>Producers per consumer</td>
<td>5.1</td>
<td>4.63</td>
</tr>
<tr>
<td>Zipf parameter</td>
<td>0.62</td>
<td>0.62</td>
</tr>
<tr>
<td>Per-producer rate</td>
<td>1/10 hour</td>
<td>1/10 hour</td>
</tr>
<tr>
<td>Zipf parameter</td>
<td>0.57</td>
<td>0.57</td>
</tr>
<tr>
<td>Per-consumer rate</td>
<td>5.8/10 hour</td>
<td>varied</td>
</tr>
<tr>
<td>Zipf parameter</td>
<td>0.62</td>
<td>0.62</td>
</tr>
</tbody>
</table>
Experiment Result: Query Rate
Experiment Result: Latency
Experiment Results: Caching
Experiment Results: CPU Load

Server

Client
Conclusions

- Consistently scale feed following
- Linear scalability
- Practical low cost solution
Thank You

• Questions?