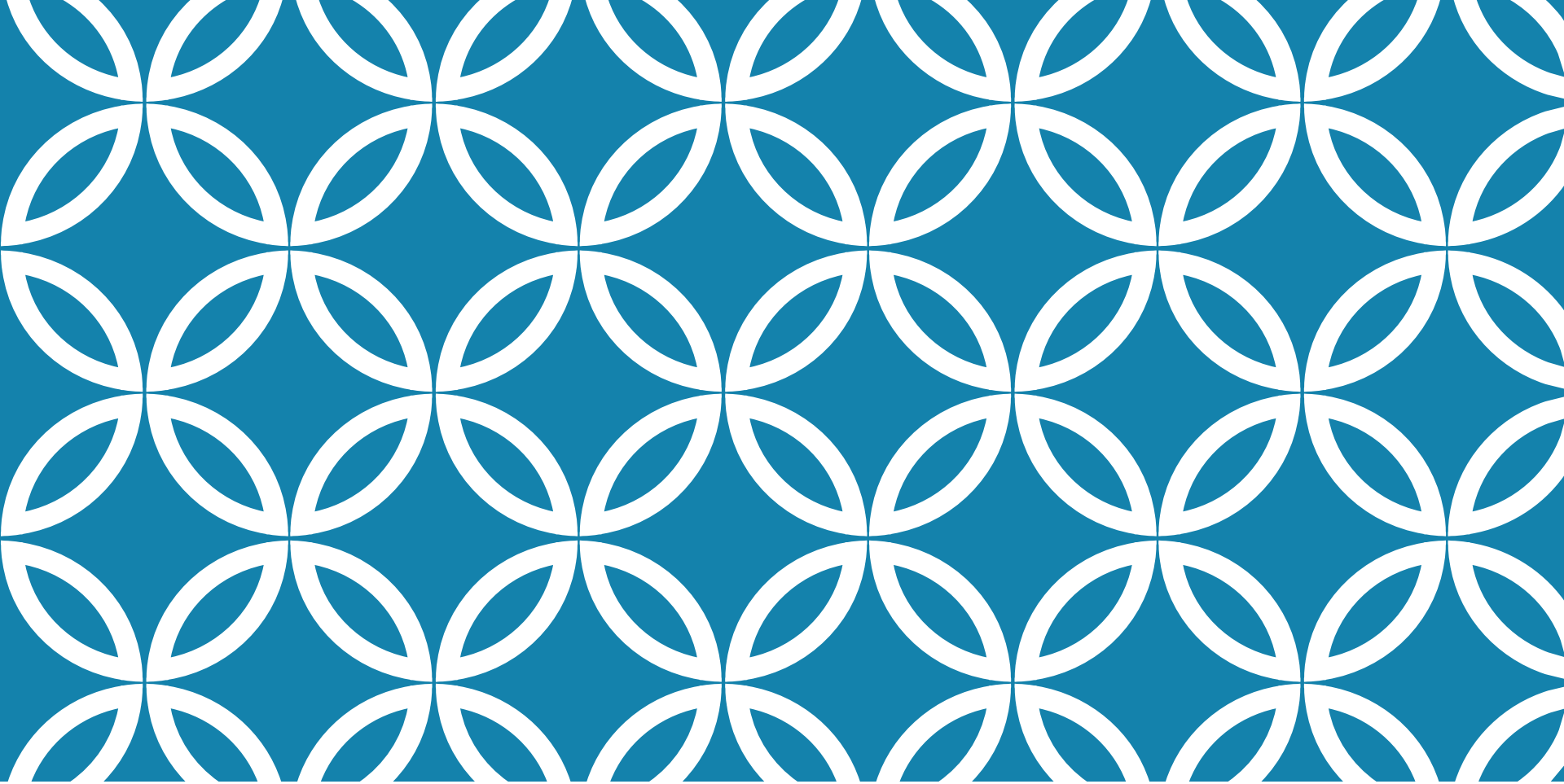
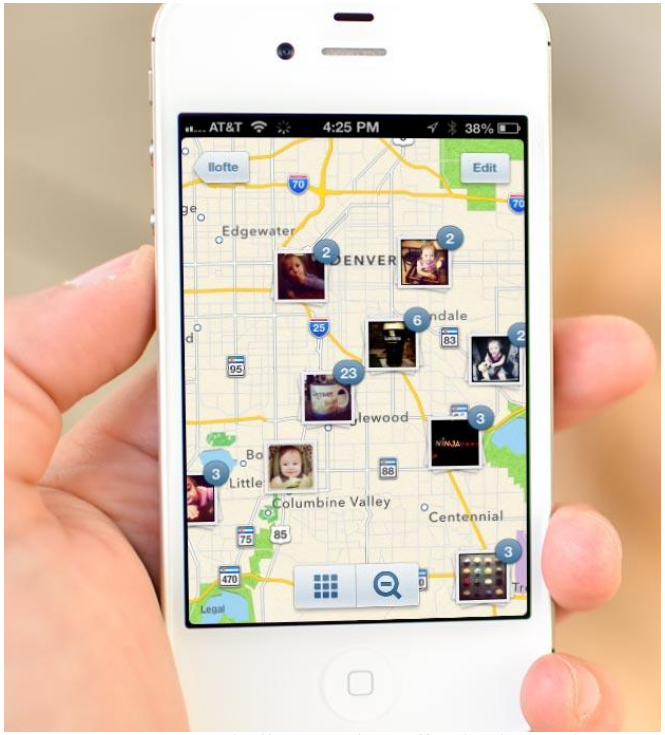


SCALING A DISTRIBUTED SPATIAL CACHE OVERLAY

Alexander Gessler
Simon Hanna
Ashley Marie Smith

MOTIVATION



http://www.imore.com/instagram-30-introduces-photo-map-and-improvements

Location-based services utilize time and geographic behavior of user

- geotagging photos
- recommendations based on user location

Heavy usage of LBS → high demand for data

> 74% of smartphone owners "use their phone to get directions or information based on current location"

Load varies temporally and spatially

- Anticipated workload – fewer queries at night
- Spatial data skew – data denser in cities

Motivation: How to adapt to loads in distributed systems characterized by spatial data?

LOAD-BALANCING

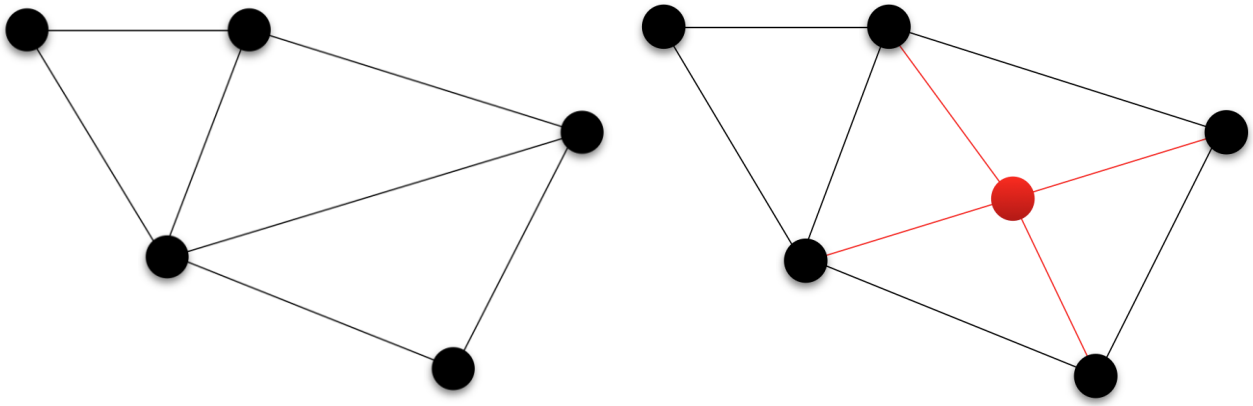
Goal: Prevent decrease in system performance caused by unexpected, heavy workloads that are unevenly distributed.

- Static
  i. Use anticipated workloads statistics

- Dynamic
  i. Reduce data skew via migration and replication
  ii. Elastic load-balancing

SCALING

- Scaling-Out: **add** resources, Scaling-In: **remove** resources

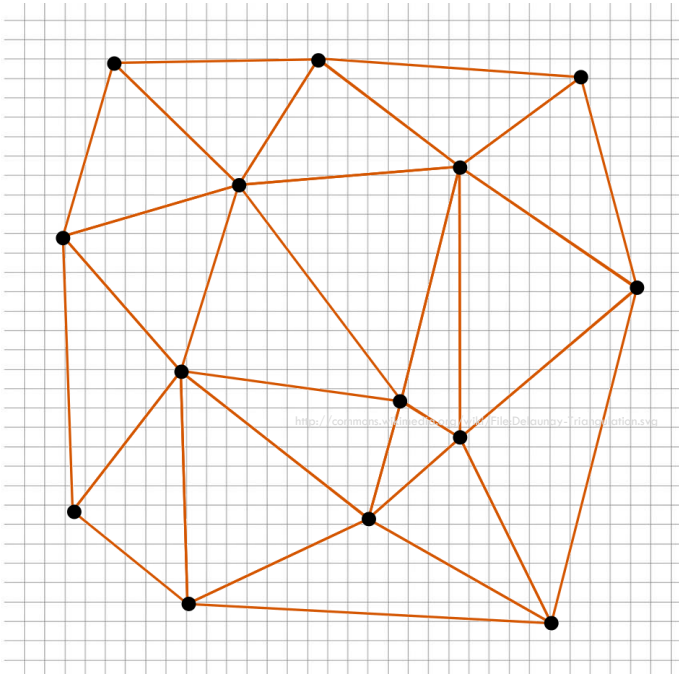


- Unlike LB: maximum system throughput changes
- Expensive: Must not use too often

OUR PROJEKT-INF CONTEXT

A grid-based cache for spatial data

- Partition spatial data into a 2D grid
- Build distributed cache overlay on top
- Topology: *Delaunay triangulation*
  Greedy forwarding / routing
- Dedicate cache nodes to cover grid partitions
  Cache focus



PROBLEM STATEMENT

We investigate whether Scaling efficiently alleviates dynamic loads in a distributed spatial cache overlay.

- Does the system scale under increasing query workload or does it need additional load-balancing mechanisms?

We quantify “efficient” by measuring response time in a real computer cluster.

- Previous study done under a simulation without looking at response time.

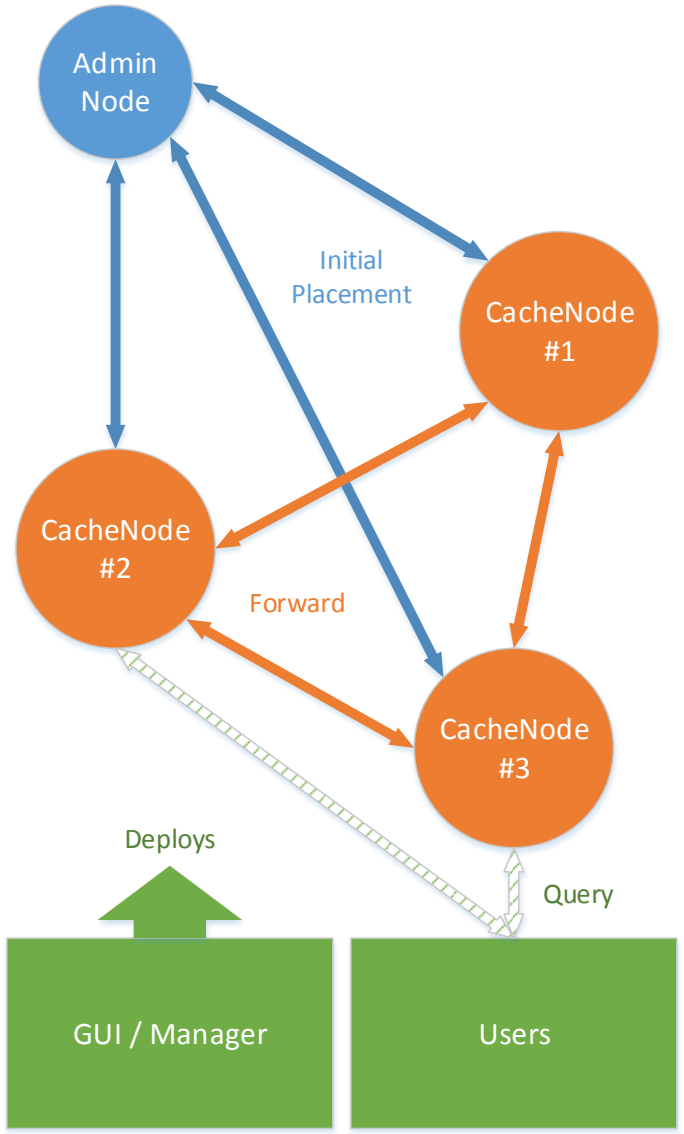
OUR PROTOTYPE

Suitable for deployment on a cluster

No real caching or query processing

We do have a “master” node (*admin*)

- Handle initial overlay construction
- Very lightweight → most tasks distributed
- Aggregate a global “load” metric



HOW WE HANDLE QUERIES

From each AWQL query a single spatial coordinate can be extracted

1. Caller sends query to an arbitrary cache node.
2. Node greedily forwards query to closest node
3. Closest node simulates processing by sleeping 50ms
4. A “Response” is send to entry node and therefrom forwarded to caller.

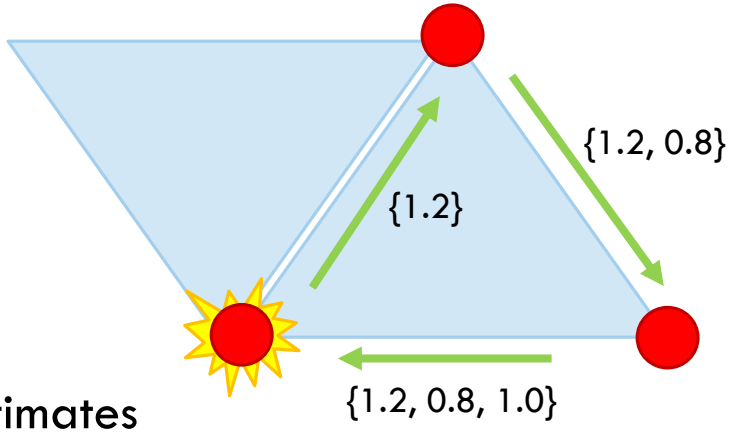
No actual caching at this time (i.e. everything is a hit).

HOW WE SCALE

High-load node periodically walks incident triangles and collects loads

If aggregate **load > threshold:**

1. Ensure mutual exclusion in area
2. Add node in triangle center
3. Have nearby nodes reset load estimates



Limitation: New cache node started locally

No Scaling-in implemented

BENCHMARK

Experiment groups

A cache overlay with **n** nodes.

Over 10 seconds, each node receives **k** queries / s. We do this for two kinds of queries:

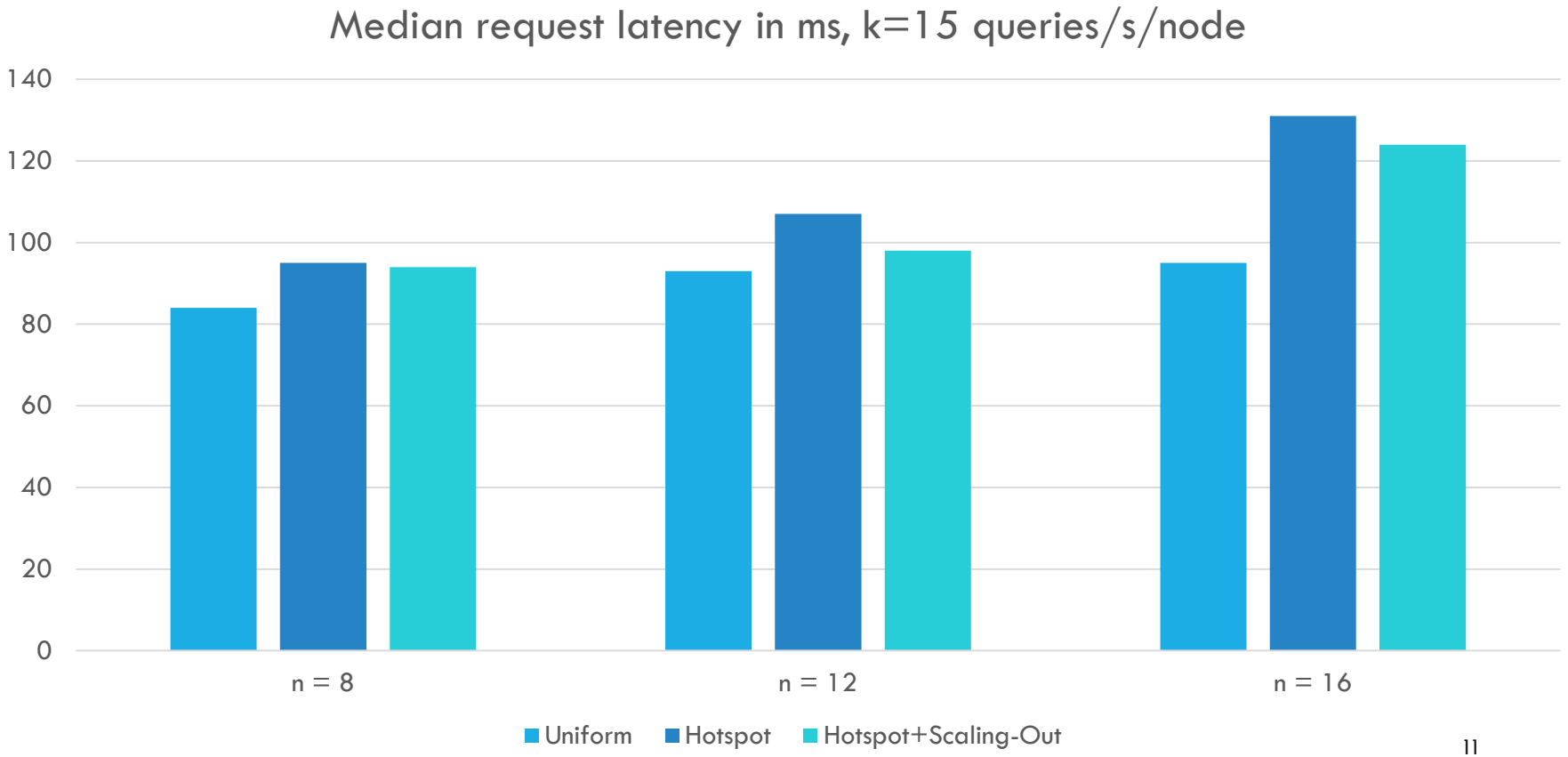
- Uniform – target coordinate is uniformly distributed across the grid
- Non-uniform – a tight Gaussian around an arbitrary point (σ = 18% grid size). Think hotspot around a big city.

We measure **median request latency**

Scaling-Out up to 200% of the initial size

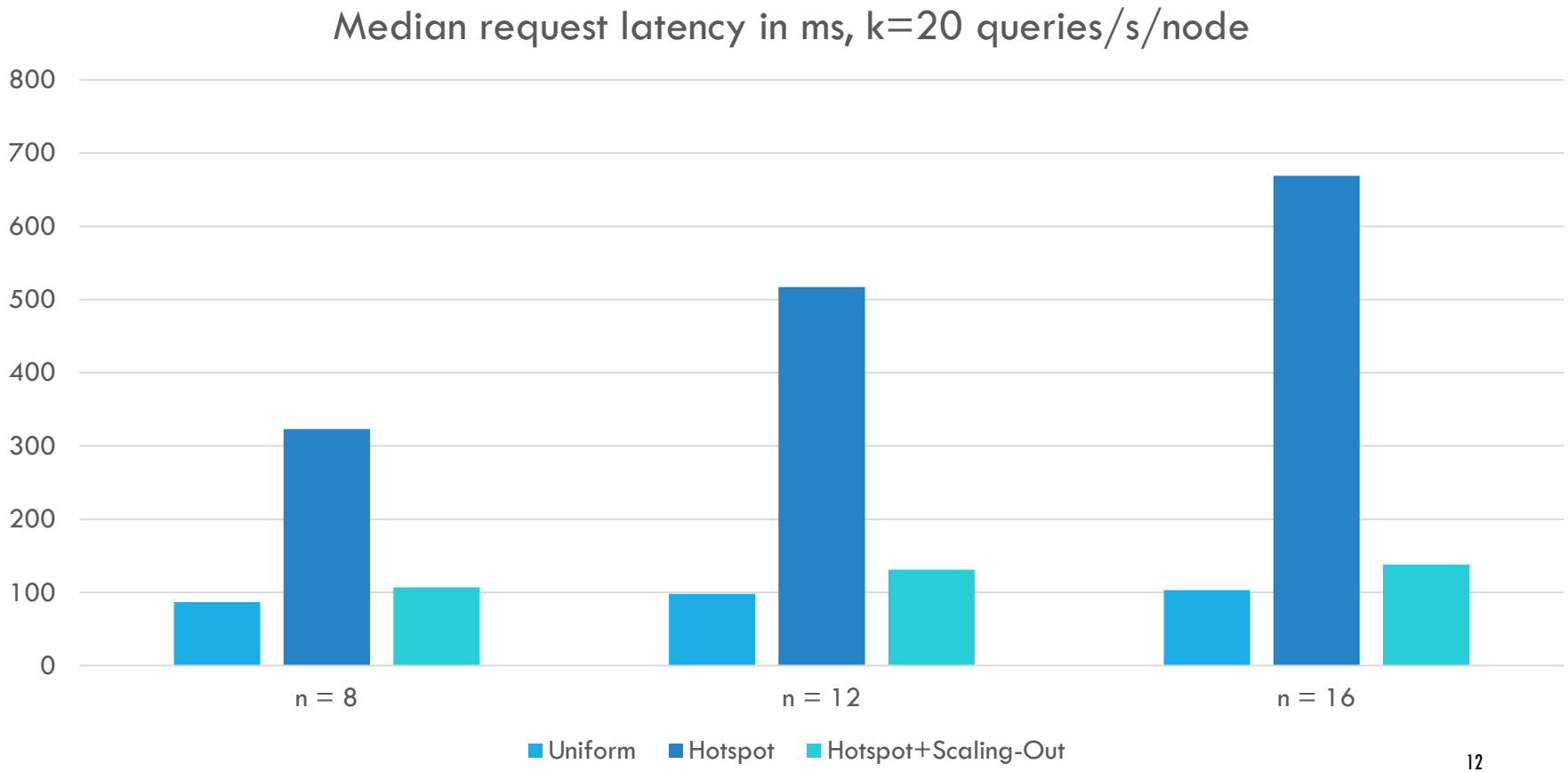
RESULTS

Average load: Scaling-Out does not make things worse, but it also does not help.



RESULTS

High load: Scaling-Out preserves quasi-linear Scalability in the non-uniform case.



CONCLUSION

Does Scaling alleviate dynamic loads in a distributed spatial cache overlay?

Yes, it does ☺

We are able to achieve median response times of about **100ms** and quasi-linear relative scalability.

But: many limitations

FUTURE WORK - SCALING

Combine Elastic Load-Balancing + Scaling

Maintain global load metric. Use Elastic LB to handle relative load differences and Scaling to respond to global load changes.

Scaling-In

Yet to be done

FUTURE WORK - ROUTING

2D Skipgraph Routing

Maintain skiplist-like probabilistic connections to 2nd, 3rd neighbors and so forth to achieve $O(log\, n)$ routing.

Local Routing Shortlist

Query messages track visited nodes + foci. As they route queries, nodes remember this free sample of the global overlay topology.

When a query enters the overlay, the entry node uses its shortlist to make the first routing decision.

TRIVIA

We contributed a paper to GI Informatiktage-2014

"Big (Data) is beautiful"

Alexander Gessler, Simon Hanna, Ashley Marie Smith, "*Scaling a distributed spatial cache overlay*", in GI Seminars Band 13 Informatiktage 2014, ISBN 978-3-885-79-447-9

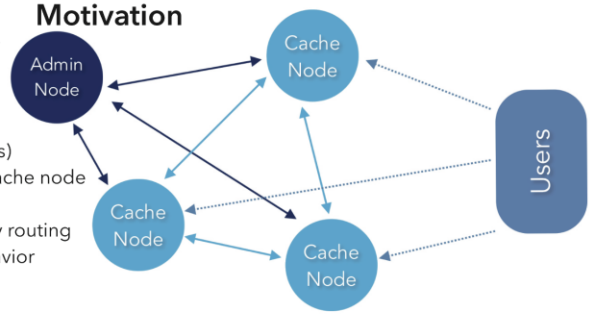
Alexander Gessler, Simon Hanna, Ashley Marie Smith
{gessleah, hannasn, smithae}@studi.informatik.uni-stuttgart.de
IPVS

Scaling a Distributed Cache Overlay

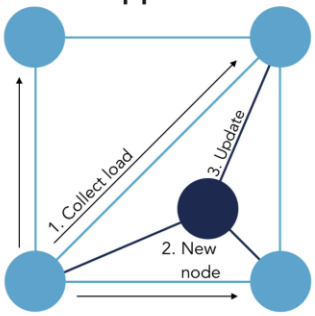
Motivation

- Cache overlay of an arbitrary number of cache nodes and one admin node
- Admin node used for initialization
- The cache overlay can be used to cache arbitrary spatial data (e.g. maps)
- Users send requests to an arbitrary cache node
- Topology is a Delaunay triangulation
- Requests are forwarded using greedy routing
- LBS utilize time and geographic behavior of users
- Load varies temporally and spatially

⇒ **How to adapt to loads in distributed systems?**



Approach



Demand-based scaling-out mechanism:

1. Collect load data from neighbors
2. Add new node in the center of existing triangle if load exceeds a certain threshold
3. Add links to update triangulation

Future Work

- Implement scaling-in
- Our current implementation exhibits a routing time of $O(\sqrt{n})$
- Use skiplists and local routing shortlists, to achieve O(log n) routing

Evaluation

Experiments:

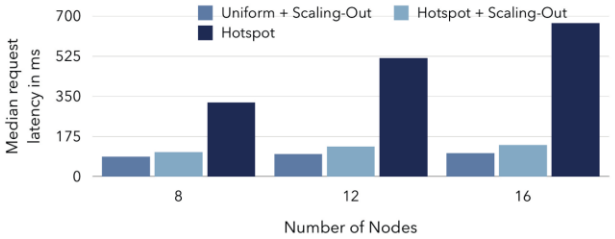
1. Uniform distribution + scaling
2. Gaussian distribution around a hotspot + scaling
3. Gaussian distribution around a hotspot

The experiments were executed with the following settings:

- 20 queries per second per node
- Each run lasts 10 seconds
- Up to 3200 requests per run

⇒ **Does the implemented system scale?**

Results:



Interpretation:

- Without scaling latency increases up to 669 ms
- With scaling, the latency decreases to 138 ms, although the workload has doubled

⇒ We confirmed that non-uniform workloads cause the system's performance and scalability to degrade under higher query rates

⇒ Using our scaling-out mechanism, we were able to scale the system's performance almost linearly in a cluster environment with sufficient bandwidth