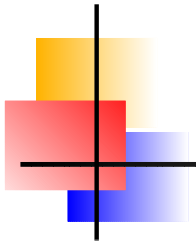
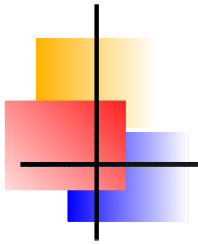


Optimizing Squid in a Filesharing Application

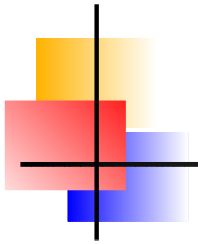
Dan Halperin

DIMACS REU 2004
with Dr. Manish Parashar
and Cristina Schmidt



Outline

- Review
  - Squid
  - Hilbert SFC
  - Modified Z-Curve
- Work
  - Hilbert Results
  - Z-Curve Results
- Future Research Areas

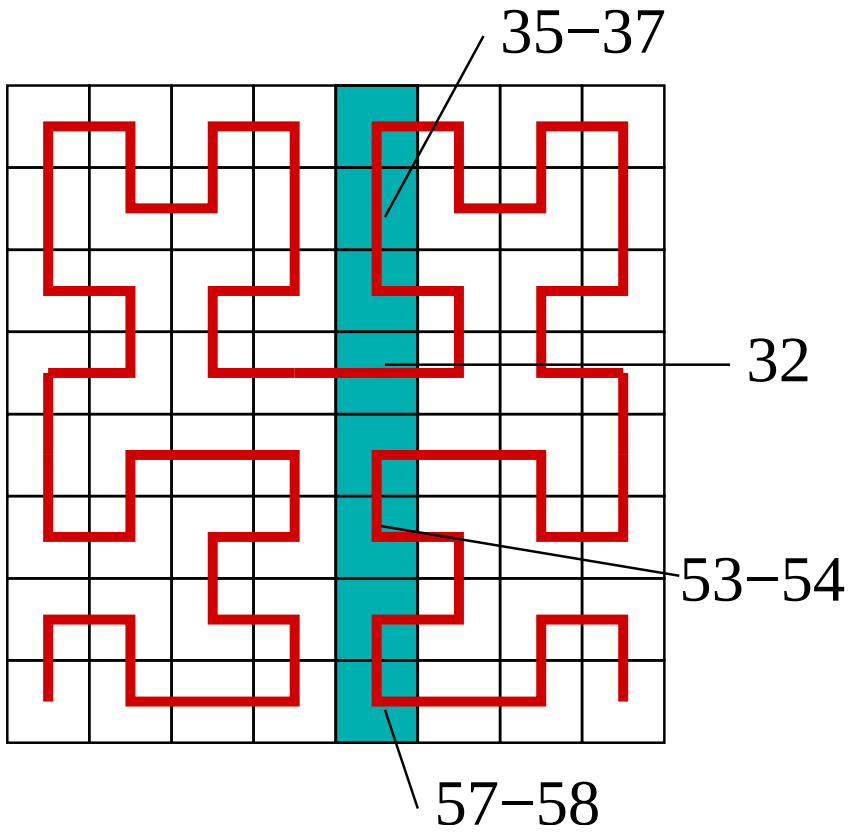


Squid P2P Model

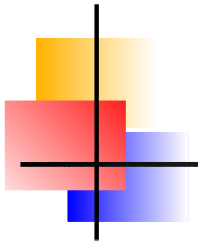
- A hash table is "a data structure that divides all elements into buckets to allow quick access to the elements." Hash function maps keys to buckets.
- Index in Distributed Hash Table (DHT).
- d keywords form a d-D keyword space $\mathcal{K}$.
- Peers given IDs in $[0, R]$ (1-D Peer ID space $\mathcal{P}$). (R = |maximum keyword| ^{d})
- Hash function $H : \mathcal{K} \rightarrow \mathcal{P}$ yields location to search for results for any query.
- Search is hashed, then DHT is queried.

Squid

Hash Function: Space-Filling Curves (SFCs)



The Hilbert SFC fills $\mathcal{K}$. Query covers some subregion of $\mathcal{K}$, and will contain some parts of the curve. The indices along the curve are the images of the hash function in $\mathcal{P}$.



Filesharing Application

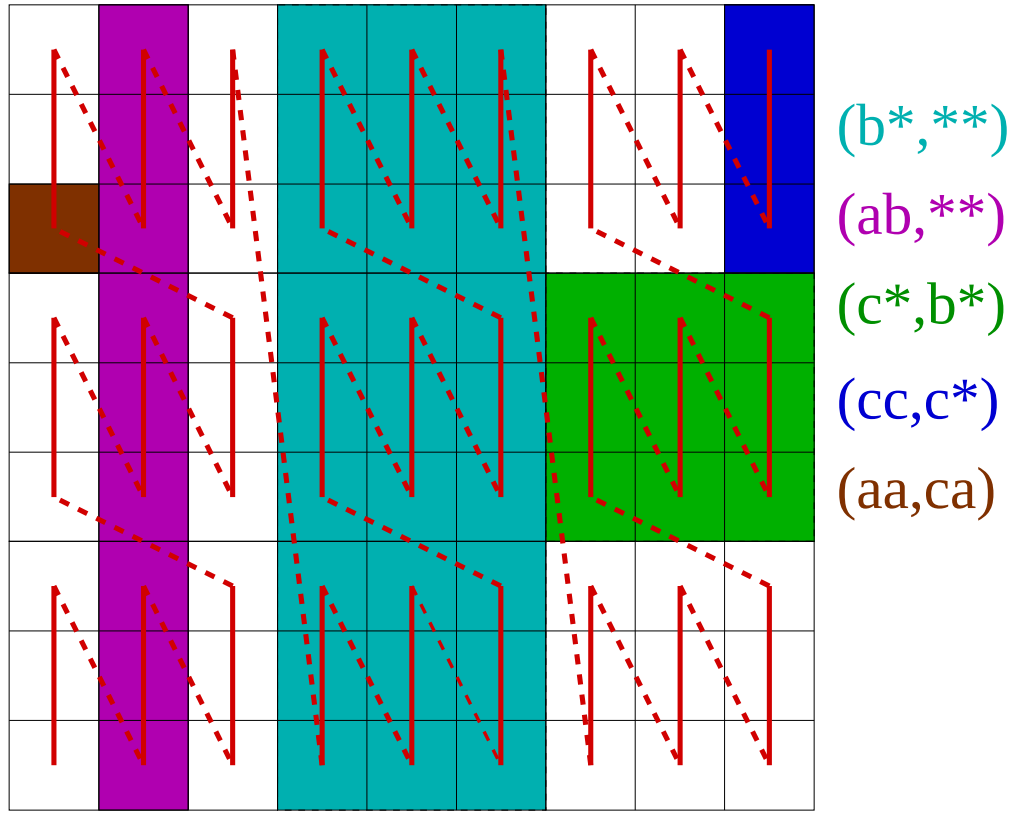
Suppose we're creating a filesharing application. Assumptions we can make:

- Interchangeable axes.
- No ranges, except for wildcards.

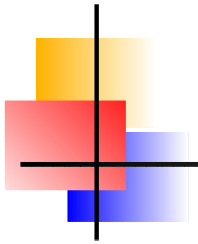
Idea: Use a modified Z-curve for the SFC.

Motivation: Do not need to preserve locality in all directions. Use simpler, more customizable Z-curve.

Sample Z SFC searches



Recall: In the Z-curve, just counting in base L. Hash function merely interleaves digits and treats as a base-L number.



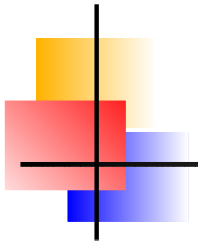
Hilbert "Results"

Speed: How fast is it?

Slow(er). In 3-D with 3 letter keywords, on the order of 40 ms / search.

Scalablity: Does the network scale well?

No. From previous simulation, beyond 3- or 4-D number of clusters becomes unmanageable and algorithm becomes too slow.

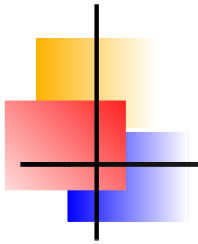


Z-Curve Simulation

Simulated network with unif. dist. peer IDs, varying

- Number of nodes: 10^2, 10^4, 10^6
- Alphabets, letters and letters + numbers.
- Dimensionality (number of keywords)
- Axis length (number of letters / keyword)

Simulation was run on my laptop (192 MB memory, 2.00 GHz Intel Celeron), i.e. not a supercomputer.



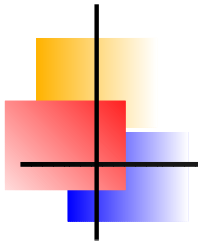
Statistics Collected

Searching:

- Number of clusters
- Number of nodes visited, relevant, % efficiency
- Messages passed, message ratio
- Elapsed time, avg. time
- Maximum search depth, "worst" time

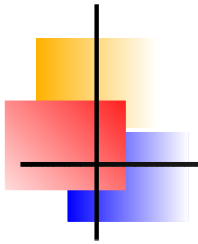
Storing data:

- Number of storing nodes
- Total time to store



Some notes

- Searches done in a L-ary parallel search fashion - refine in parallel digit by digit.
- When storing data, since axes are permutable, must be stored in $d!$ indices. $10! > 3 \times 10^6$. But, can use pruning to skip permutations that would be stored at the same node.
- Use least-specific queries for worst search results. (j*,*,...,*)
- Fix keyword length at 8. Variation here has almost no change.

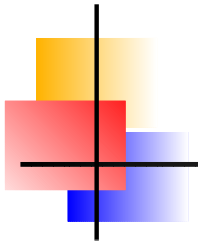


So is it scalable?

Average time (ms)

L	D	10^2	10^4	10^6
26	3	?	0.56	0.80
	5	?	0.89	1.27
	7	?	1.24	1.76
	10	?	1.86	2.60
36	3	?	0.41	0.84
	5	?	0.66	1.28
	7	?	0.88	1.82
	10	?	1.32	2.68

Looks like yes! But wait...



So is it scalable?

Message Ratio

L	D	10^2	10^4	10^6
26	*	1.0	1.30	1.46
36	*	1.0	1.14	1.36

Ok, still looks like yes. What about

Efficiency? Goes to 100% as any parameter increases

Search Depth? Constant maxes around 6 for any parameters

 # Clusters? Blows up! But it seems irrelevant.

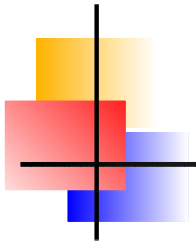
All right. The search part is scalable!

What about indexing data?

Percent of Nodes to Store (a,c,e,g,i,k,m,o,s)

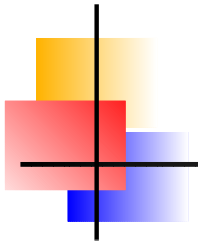
L	D	10^2	10^4	10^6
26	3	4%	0.06%	0.0006%
	5	10%	0.70%	0.012%
	7	18%	2.46%	0.14%
	10	36%	8.64%	1.19%
36	3	3%	0.06%	0.0006%
	5	8%	0.48%	0.012%
	7	14%	1.43%	0.096%
	10	23%	4.25%	0.61%

Look good, but *Where are the factorials?!*



Future Work

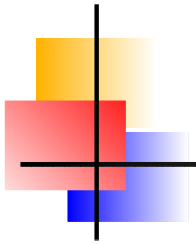
- Actual Hilbert data?
- How about some overhead accounting?
- Fix that uniform distribution! (Load balancing)
- What do the ratios mean?
- What happened to the factorials?!
- How much *does* clustering matter, anyway?
- Can we prove anything?



References

(1) C. Schmidt and M. Parashar, "Enabling Flexible Queries with Guarantees in P2P Systems," *IEEE Internet Computing* (May-June 2004), pp.19-26.

(2) C. Schmidt and M. Parashar, "Peer to Peer Information Storage and Discovery Systems," forthcoming.



Acknowledgments

- Dr. Manish Parashar
- Cristina Schmidt

- DIMACS REU program
- Rutgers University
- Vincent Matossian