Squirrel: A peer-to-peer web cache

Sitaram Iyer (Rice University)

Joint work with
Ant Rowstron (MSR Cambridge)
Peter Druschel (Rice University)

PODC 2002 / Sitaram Iyer / Tuesday July 23 / Monterey, CA

Pizza talk at CS Rice. Feb 25, 2002.

PODC 2002 / Tuesday July 23 / Monterey, CA.

Web Caching

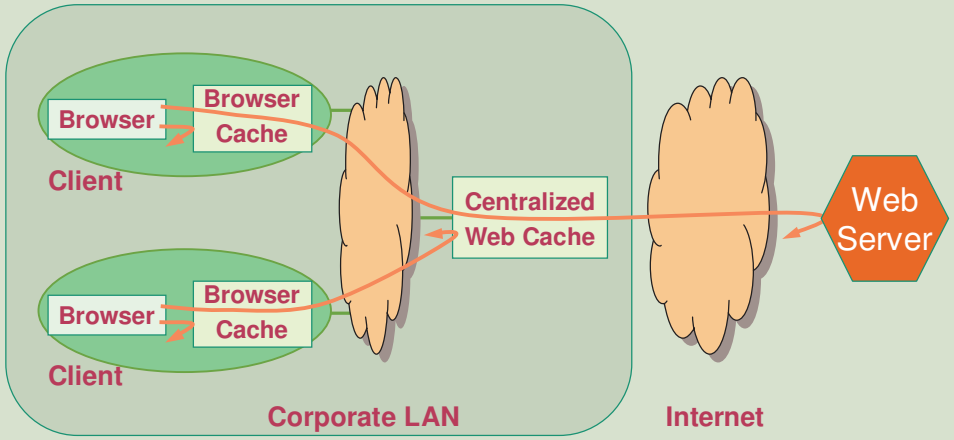
1. Latency,
2. External traffic,
3. Load on web servers and routers.

Deployed at: Corporate network boundaries, ISPs, Web Servers, etc.

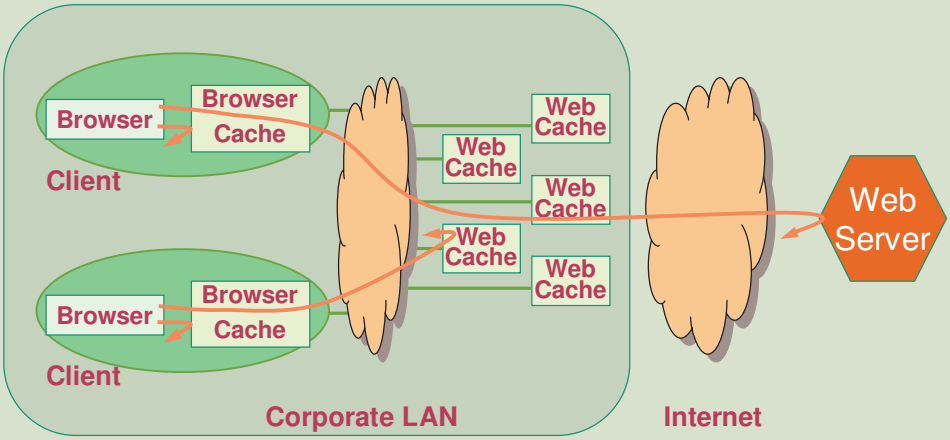
Object = html page, image, etc.

Focus on corporate network boundary caches.

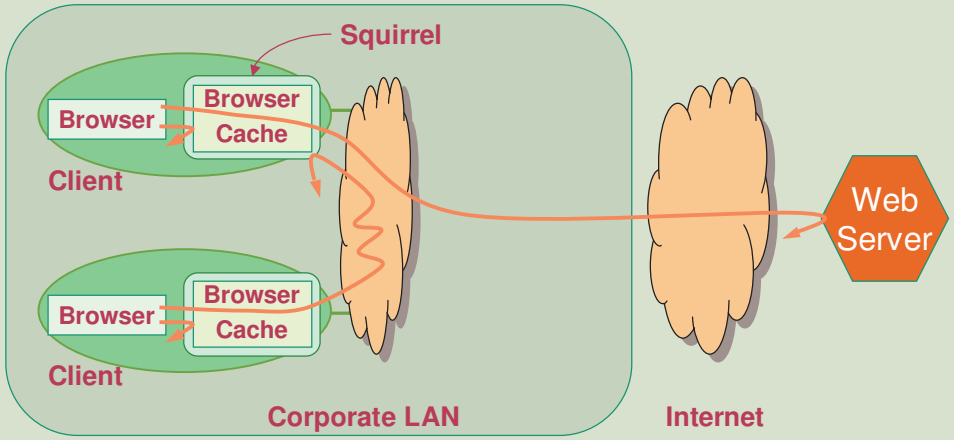
Web Cache


Browser
Browser
Cache
Client
Browser
Browser
Cache
Client
Centralized
Web Cache
Corporate LAN
Internet
Web
Server

Cooperative Web Cache

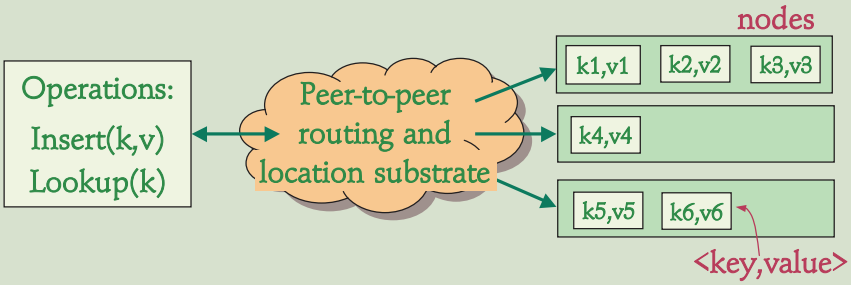


Decentralized Web Cache



Distributed Hash Table

Peer-to-peer location service: Pastry



- Completely decentralized and self-organizing
- Fault-tolerant, scalable, efficient

Why peer-to-peer?

1. Cost of dedicated web cache
   - No additional hardware
2. Administrative effort
   - Self-organizing network
3. Scaling implies upgrading
   - Resources grow with clients

Dedicated hardware. WHY? In some large organizations in the north-west, they use clusters of as many as 30 machines. Overprovision resources for peak loads, why?

Administrative costs, both in terms of configuring those machines, and for cooperative web caches, setting up the hierarchy or mesh or whatever. Why? Much nicer to have it self-configuring; typically there is some software dissemination mechanism, so installing is like a few clicks, and done.

Constant pressure for upgrading the cluster. With p2p, number of "clients" = number of "servers", so it is potentially self-scaling.

Many people don't use web caches because they fail and their connections go bang. In this model, so what if some nodes die?!

Again, these are only potential benefits. They do not automatically happen.

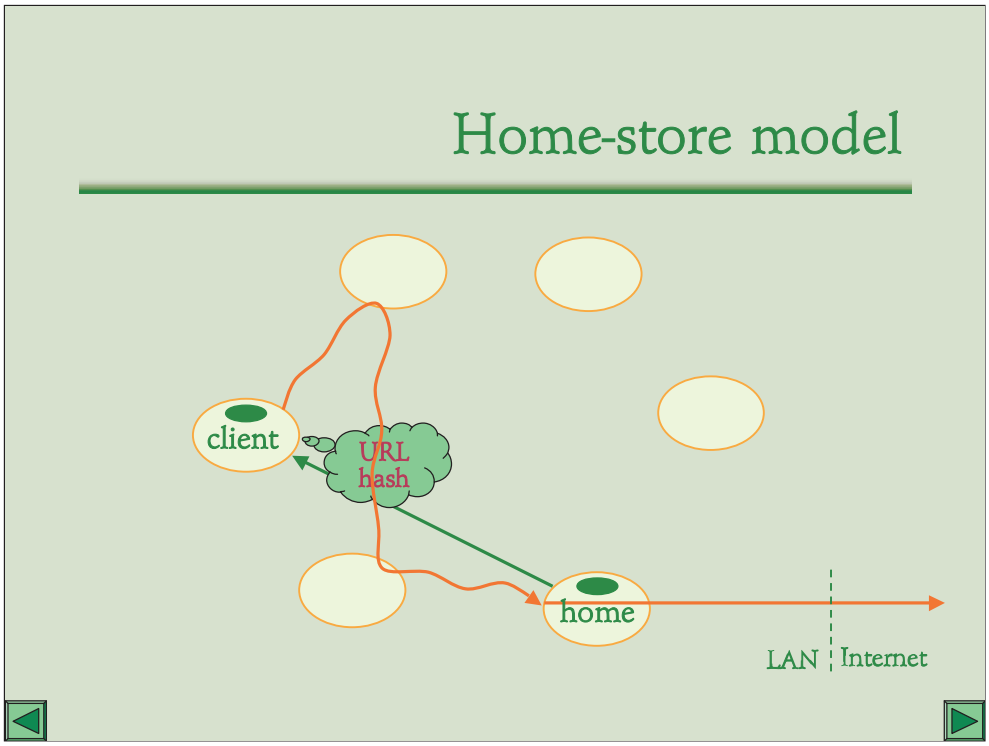
Setting

- Corporate LAN
- 100 - 100,000 desktop machines
- Located in a single building or campus
- Each node runs an instance of Squirrel
- Sets it as the browser's proxy

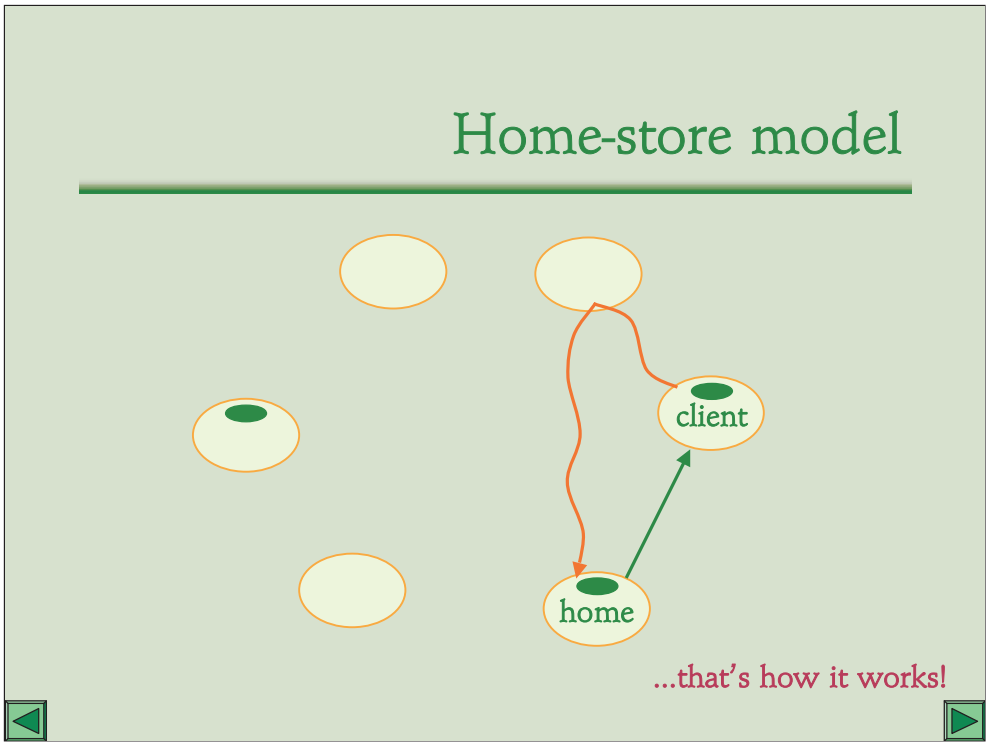
Mapping Squirrel onto Pastry

Two approaches:

- Home-store
- Directory



Now that we have a routing protocol, I'll propose two schemes for mapping Squirrel onto it.



Verry simple.

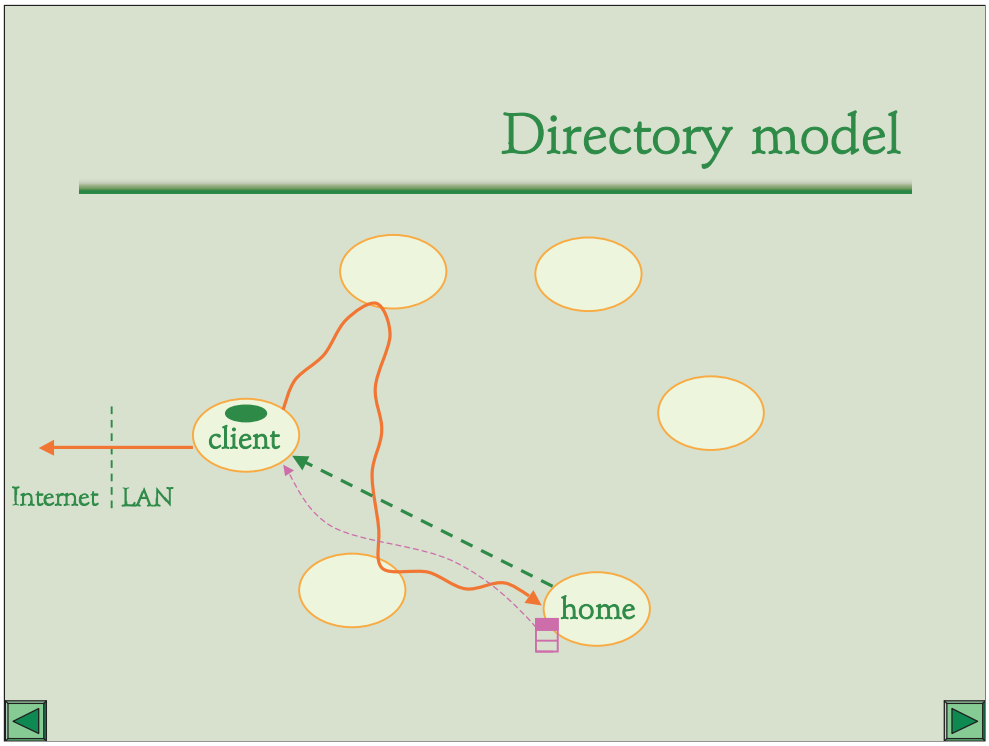
Directory model

Client nodes always cache objects locally.

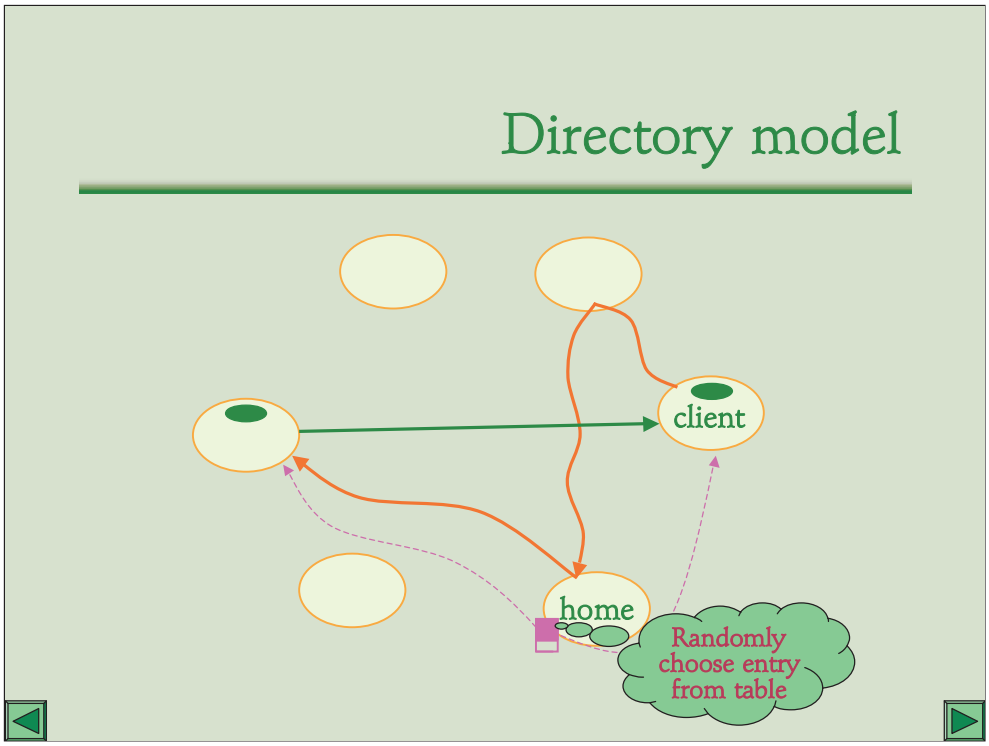
Home-store: home node also stores objects.

Directory: the home node only stores pointers to recent clients, and forwards requests.

Objects are always stored at clients; in the home-store model, they are stored at home nodes too. Why? Suppose the home node does not store the object, but merely maintains a pointer to nodes that recently accessed the object, and forwards subsequent requests to these nodes.



Home says “go get it yourself”. … Meanwhile … creates a directory and stores a pointer to this client. By pointer I mean, of course, the IP address.



(explain protocol). What's the motivation? One is that we don't store objects at the home. But more interestingly, we expect that for an object accessed by many nodes, the directory of most recently accessing nodes keeps rapidly changing. So a randomly chosen entry will achieve some kind of load balancing. Whether this happens, we'll see.

Directory: Advantages

Avoids storing unnecessary copies of objects.

Rapidly changing directory for popular objects seems to improve load balancing.

Home-store scheme can incur hotspots.

Appreciate that these two designs are the two endpoints of the design space, based on what choice you make regarding object storage location.

If an object is in the centralized cache, it should be on some client node in both p2p caching schemes; so we expect, roughly, that hit ratio is the same. User-perceived latency is overshadowed by accesses outside the network, so if the hops within the LAN aren't too many, then all methods are roughly similar.

Directory: Disadvantages

Cache insertion only happens at clients, so:

- active clients store all the popular objects,
- inactive clients waste most of their storage.

Implications:

1. Reduced cache size.
2. Load imbalance.

There's a strange quirk in the directory scheme. … … When some node later accesses this html page with many images, all these requests pounce on a single client, and its load shoots. Does this matter? We don't know; simulations will tell.

Directory: Load spike example

- Web page with many embedded images, or
- Periods of heavy browsing.

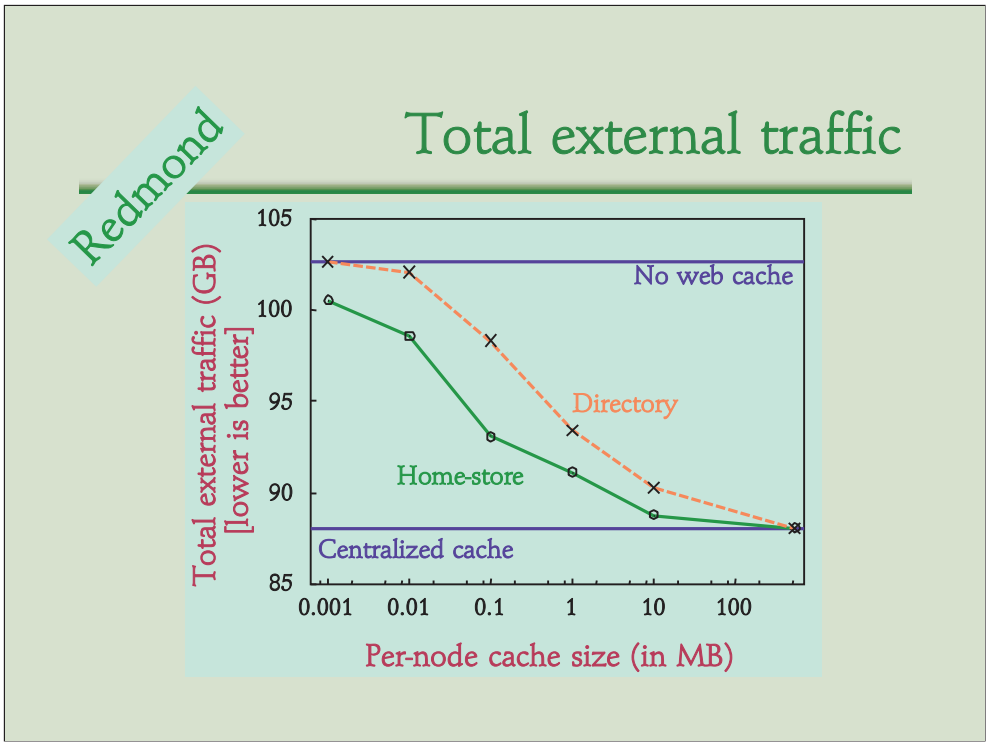
Many home nodes point to such clients!

Evaluate …

Trace characteristics

Microsoft in :	Redmond	Cambridge
Total duration	1 day	31 days
Number of clients	36,782	105
Number of HTTP requests	16.41 million	0.971 million
Peak request rate	606 req/sec	186 req/sec
Number of objects	5.13 million	0.469 million
Number of cacheable objects	2.56 million	0.226 million
Mean cacheable object reuse	5.4 times	3.22 times

Two traces with very different characteristics, and hoping to bring out the fundamental properties of the protocols. All clients act as Squirrel nodes. Wide variation in number of requests. Peak request rate high, so we need a cluster of machines (or a powerful machine in the latter case) for a centralized cache. About half the objects are cacheable; others are sent out by the client directly. There is some reuse of cacheable objects; otherwise web caching would be pointless.



/* First metric, external bandwidth. Define.

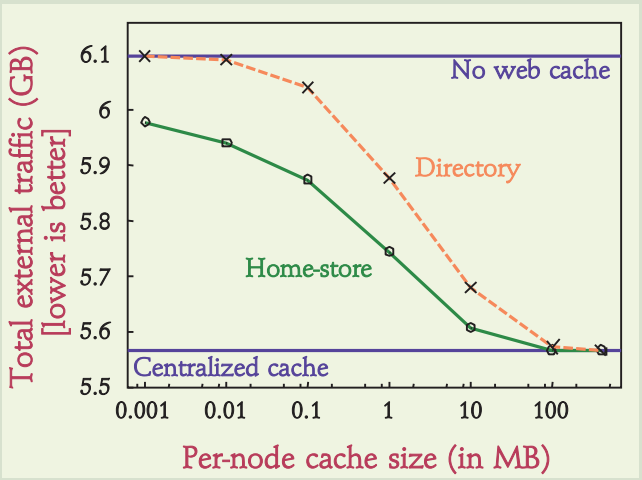
1. Two blue lines depict extbw with a centralized cache of infinite size and no cache; this difference is the benefit of web caching in terms of extbw.
2. X-axis is logscale; even for small values of per-node cache, there is performance close to centralized. This shows that there is good pooling of disk space from around the network.
3. Home-store performs better than directory. This is because some node in the latter stores a large page with many images, or many web pages; when it fills up and evicts something, the subsequent clients need to go out. Home-store's natural load balancing helps avoid hotspots of storage. This is interesting, because although home-store stores more, its storage *utilization* is significantly better than directory. */

New:

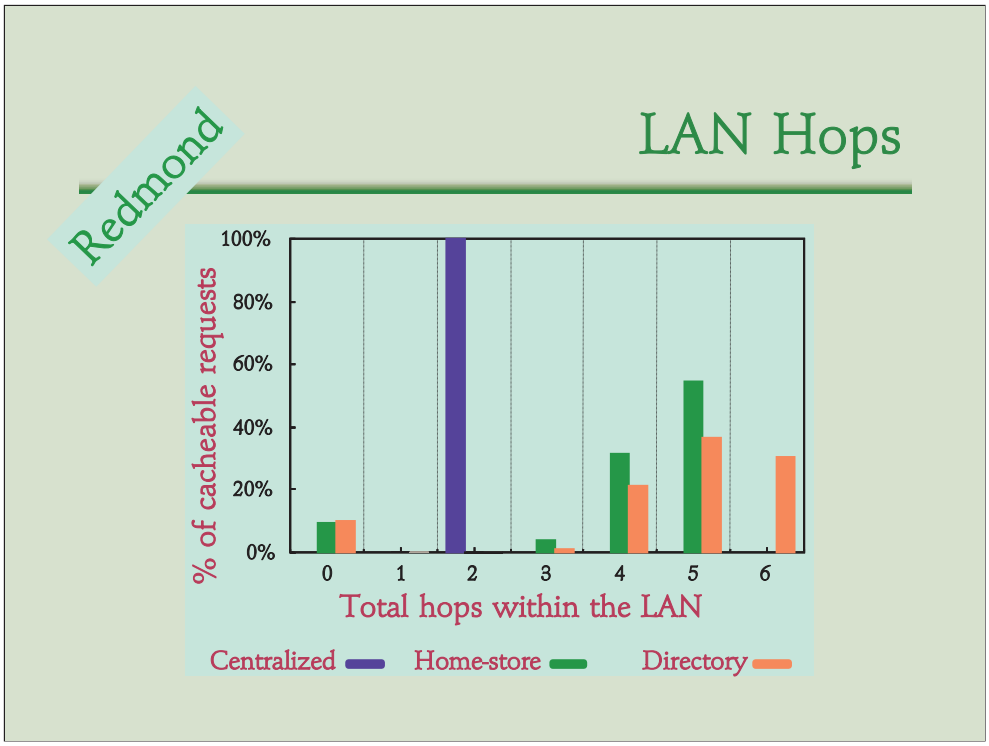
This graph depicts the total data traffic to the organization, in GB over the period of the entire trace. The two blue lines represent the cases for no web cache, and a centralized cache with sufficient disk capacity; the difference is the benefit due to web caching in terms of external traffic. The x-axis is the storage contributed per node in MB. Two key observations here. Firstly, with as little as about 100MB contributed by each node, Squirrel performs comparable to a centralized web cache in terms of external traffic. Secondly, home-store is better than directory, even though home-store stores more. This is because home-store distributes objects across nodes randomly, whereas directory relies on clients for storage. So an active client caches a lot of popular objects, whereas an inactive client essentially wastes its disk

Cambridge

Total external traffic



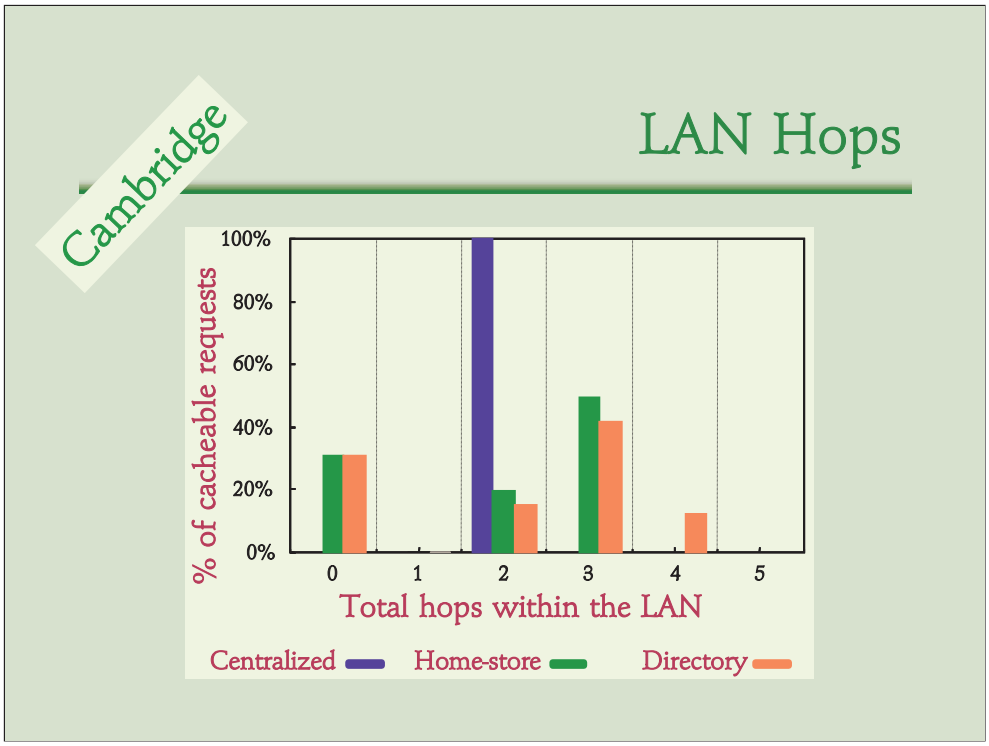
Similar behaviour, only scaled down in extbw. Again, Squirrel is as good as a Centralized cache for about 100MB of contribution.



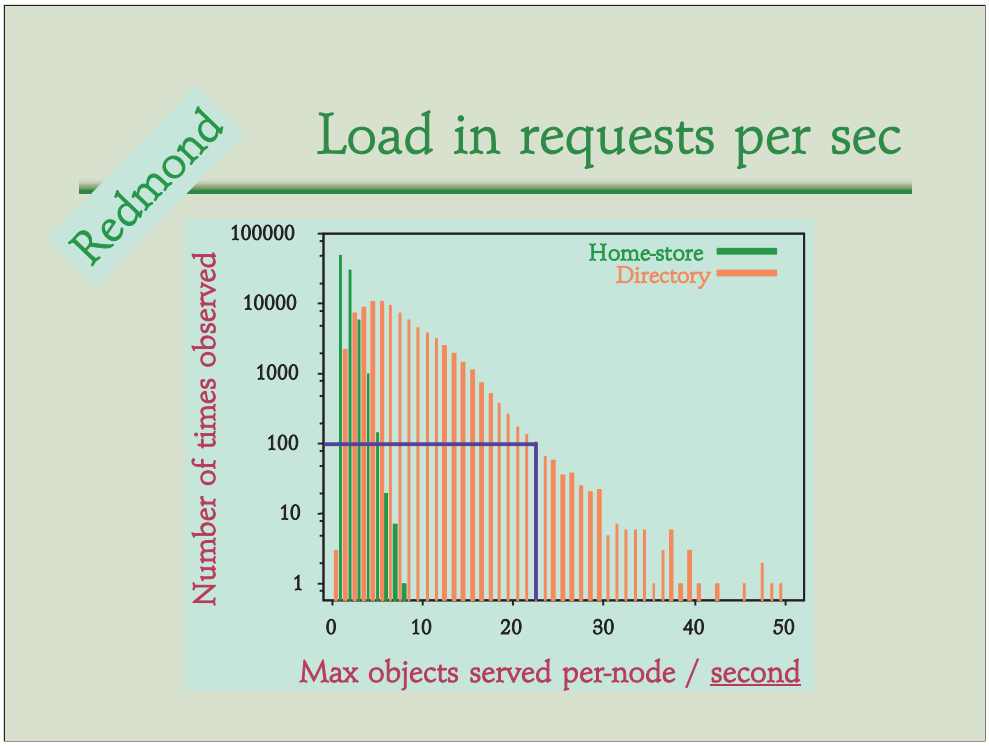
/* Latency is overshadowed by external accesses *if* LAN hops are few (not like 50). Centralized to and fro. Home-store 3-to-4 + 1. Directory +1 sometimes when it forwards; when home node says "go get it yourself" there is no forwarding. Very rare occasions when delegate has failed and two more hops. */

New:

The other benefit of web caching is to improve client latency. Notice that Squirrel and a centralized cache use the same cache expiration policy, so they achieve the same hit rate given enough disk storage. So requests that miss in Squirrel also miss in the Centralized cache and achieve the same latency. Requests that hit in the cache have latency corresponding to the number of LAN hops, and that's this graph. Under the corporate LAN assumption, internal latency is small compared to external latency, provided there are not too many hops within the LAN. That's being depicted here; even for a 36000 node network, Squirrel only takes 3-4 hops to get to the home node, possibly one forwarding hop for directory, and one hop for the return path. This is basically comparable to the two hops – to and fro – in the centralized cache.



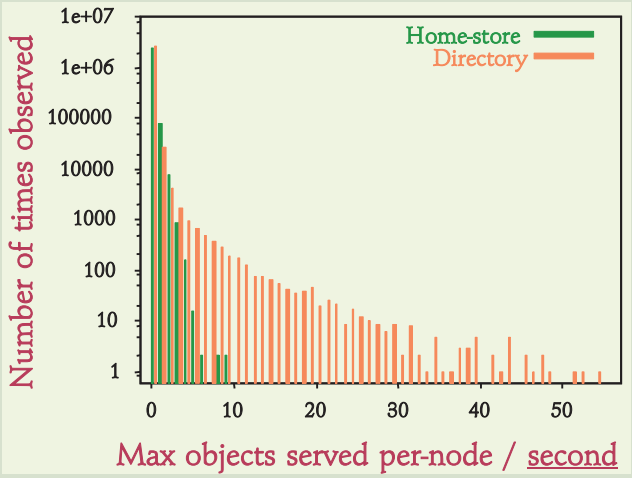
Same thing, only shifted left: Home-store 1-to-2 hops + 1.



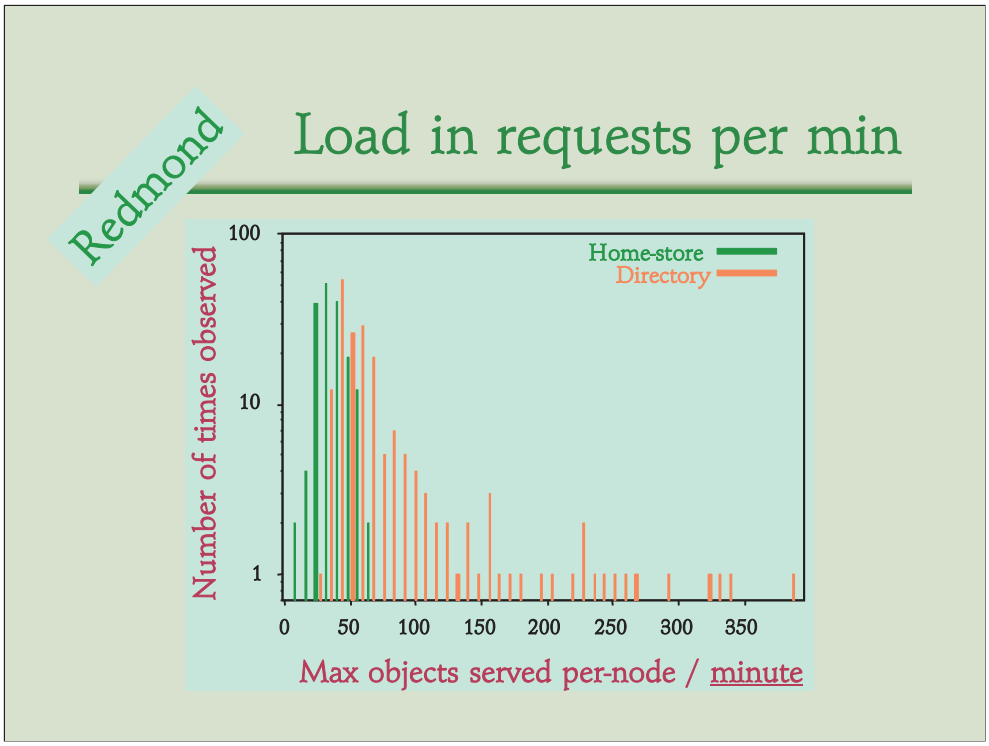
Finally, load. This is a problem that isn't there with a centralized cache, that we need to maintain bursty and sustained load low. <click> Consider this point: it means that there are a hundred occasions during the entire trace when some node in the network services as many as 23 requests in some second. So a narrow, left-stacked set of bars is a good thing. Means there is one occasion during the day when some node services 50 requests/sec: which is a tad on the high side. Home-store's natural load balancing keeps load always as low as 10 req/sec.

Cambridge

Load in requests per sec



Fact that both peak loads are almost the same numbers as in the Redmond trace speaks for the scalability of the system.

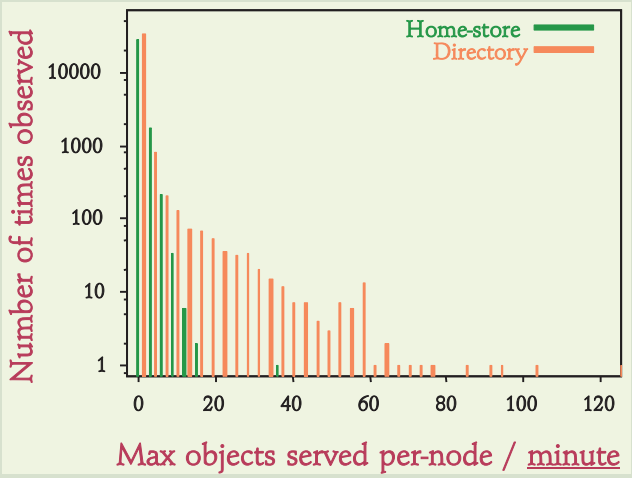


We talked of sustained load; it can be illustrated by measuring per-minute load. 370 requests per minute is rather high, compared to 60 per minute.

Recall we talked of two reasons for the directory protocol quirk: a page with many images, and a previously heavily browsing client. The former can result in a few-seconds-long burst of requests, but can hardly be expected to sustain 370 requests for a minute. So the latter reason is important too.

Cambridge

Load in requests per min



Similar.

Fault tolerance

Sudden node failures result in
partial loss of cached content.

Home-store:	Proportional to failed nodes.
Directory:	More vulnerable.

Fault tolerance

If 1% of Squirrel nodes abruptly crash, the fraction of lost cached content is:

	Home-store	Directory
Redmond	Mean 1% Max 1.77%	Mean 1.71% Max 19.3%
Cambridge	Mean 1% Max 3.52%	Mean 1.65% Max 9.8%

Conclusions

- Possible to decentralize web caching.
- Performance comparable to a centralized web cache,
- Is better in terms of cost, scalability, and administration effort, and
- Under our assumptions, the home-store scheme is superior to the directory scheme.

Moral: the simpler Home-store scheme beats the complicated Directory scheme hands-down.

Other aspects of Squirrel

- Adaptive replication
  - Hotspot avoidance
  - Improved robustness
- Route caching
  - Fewer LAN hops

Thanks.

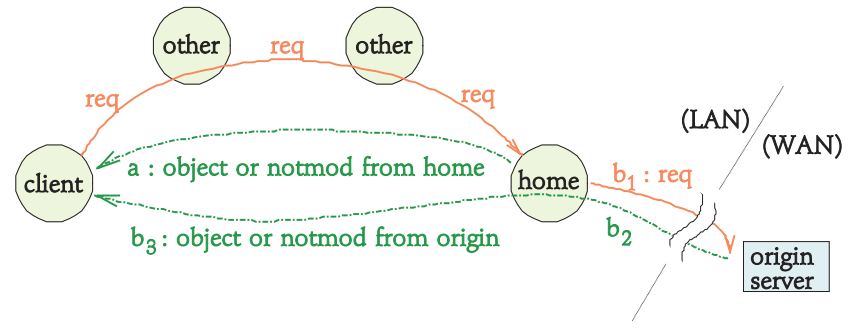
(backup) Storage utilization

Redmond	Home-store	Directory
Total	97641 MB	61652 MB
Mean per-node	2.6 MB	1.6 MB
Max per-node	1664 MB	1664 MB

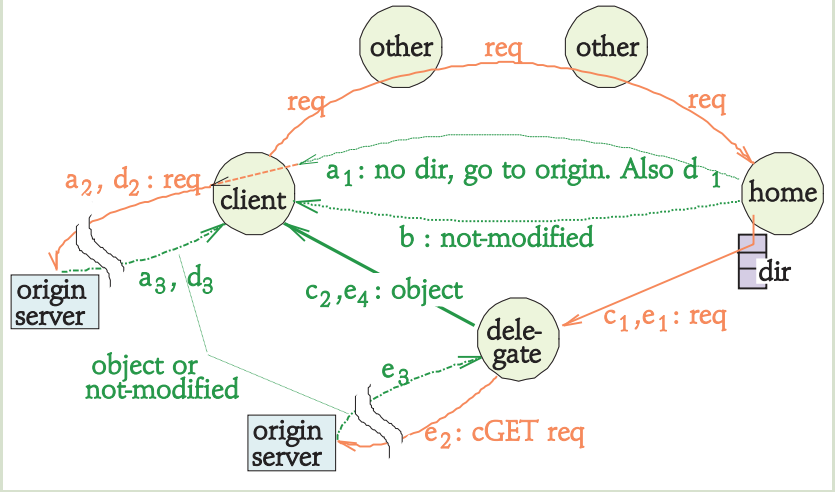
(backup) Fault tolerance

	Home-store	Directory
Equations	Mean H/O Max H_{max}/O	Mean (H+S)/O Max $max(H_{max}, S_{max})$/O
Redmond	Mean 0.0027% Max 0.0048%	Mean 0.198% Max 1.5%
Cambridge	Mean 0.95% Max 3.34%	Mean 1.68% Max 12.4%

(backup) Full home-store protocol



(backup) Full directory protocol



(backup) Peer-to-peer Computing

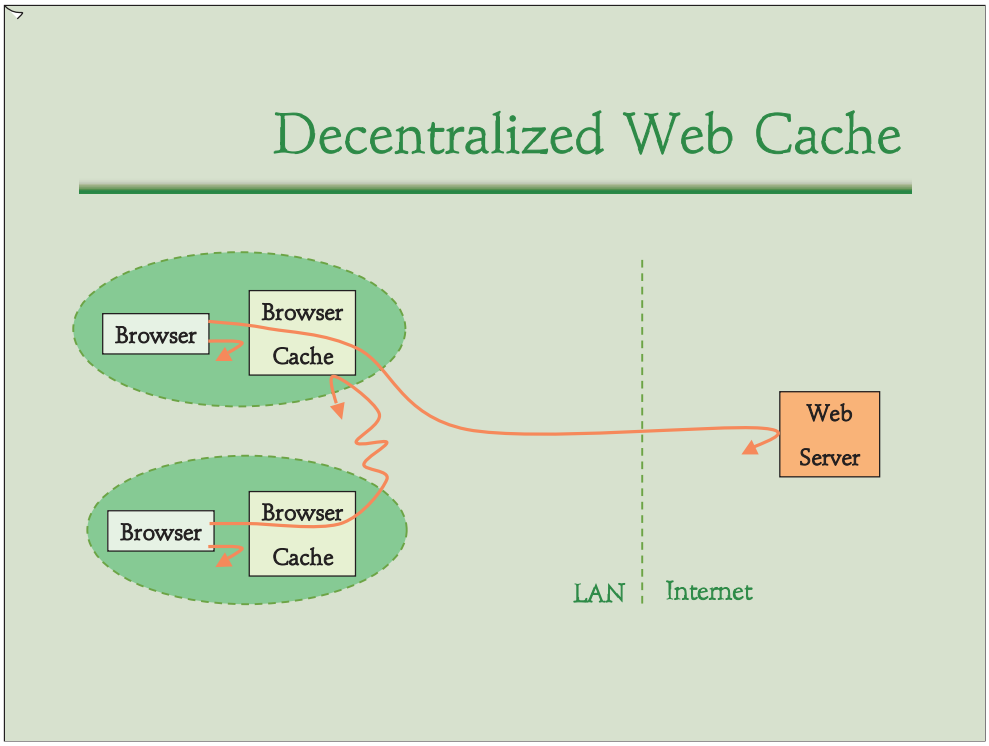
Decentralize a distributed protocol:

- Scalable
- Self-organizing
- Fault tolerant
- Load balanced

Not automatic!!

P2P is a distributed computing paradigm where protocols are decentralized, all nodes do the same thing, and somehow the entire network collaborates to perform the desired function. The trick is to decentralize your standard distributed protocol, such as network file storage, event notification, anonymization, etc. It becomes scalable, … (expand).

Curious to note that these benefits do not come automatically; the protocol has to be decentralized just right. In this talk, I'll try to give a flavour of this using the specific problem of web caching.



The goal of this talk is to get rid of this box. When a node doesn't have an object in your local cache, it "somehow" discovers that a different node has it, and gets it from there. Kinda like a cooperative cache. So these nodes export their local caches to other nodes in the network, and these combine to form a large virtual cache.

Challenge

Decentralized web caching algorithm:

- Need to achieve those benefits in practice!
- Need to keep overhead unnoticeably low.
- Node failures should not become significant.

Peer-to-peer routing, e.g., Pastry

Peer-to-peer object location and routing substrate = Distributed Hash Table.

Reliably maps an object key to a live node.

Routes in $\log_{16}(N)$ steps

(e.g. 3-4 steps for 100,000 nodes)

If we want to go about constructing a p2p cache, the easiest way, I believe, is to leverage a p2p routing protocol. For instance, Pastry. The fancy name is a p2p object location and routing substrate; what it really gives you is distributed hash table functionality. (explain)

Home-store is better!

Simpler home-store scheme achieves load balancing by hash function randomization.

Directory scheme implicitly relies on access patterns for load distribution.

Directory scheme seems better...

Avoids storing unnecessary copies of objects.

Rapidly changing directory for popular objects results in load balancing.

Appreciate that these two designs are the two endpoints of the design space, based on what choice you make regarding object storage location.

If an object is in the centralized cache, it should be on some client node in both p2p caching schemes; so we expect, roughly, that hit ratio is the same. User-perceived latency is overshadowed by accesses outside the network, so if the hops within the LAN aren't too many, then all methods are roughly similar.

Interesting difference

Consider:

- Web page with many images, or
- Heavily browsing node

Directory: many pointers to some node.

Home-store: natural load balancing.

Evaluate ...

There's a strange quirk in the directory scheme. … … When some node later accesses this html page with many images, all these requests pounce on a single client, and its load shoots. Does this matter? We don't know; simulations will tell.

Fault tolerance

When a single Squirrel node crashes, the fraction of lost cached content is:

	Home-store		Directory	
Redmond	Mean	0.0027%	Mean	0.2%
	Max	0.0048%	Max	1.5%
Cambridge	Mean	0.95%	Mean	1.7%
	Max	3.34%	Max	12.4%