Optimizing Chord as a Distributed Cache

29th March 2004

1 The Problem

A reliable peer-to-peer network for content delivery could be extremely helpful for the functioning of the internet. The concept of a "mirror" might disappear as every piece of static content becomes automatically mirrored. The "slashdot effect" (a sudden focus of attention on a rarely used website resulting in the website being disrupted) may be greatly ameliorated or removed. These are effects of significant economic impact as companies (Akamai and Cisco for example) have several products dedicated to making individual pieces of content more widely available.

Instead of mirrors or proprietary products, we might imagine that an overloaded website starts returning the hash of a desired document rather than the document. As a hash is only 128 bits long, servicing this request requires very little work by the server or the network connected to the server, allowing very high service loads using little bandwidth.

Of course, the hash is not the document that the client desires. In order to transform the hash into a document, we would like a fast distributed index into the caches of other clients. We focus on using a variant of Chord [1] to provide this infrastructure. Next, we discuss two mostly orthogonal modifications of chord which may be helpful in optimizing it as a webcache.

1. Making a virtual node for every document.
2. Using a distributed nearest-neighbor algorithm to optimize network latency.

We discuss the modifications and then present the combined algorithm.

2 A Virtual Node for every Document

This is an easy to describe modification of the standard chord protocol: Make every document be a virtual node. The modification is mathematically appealing because it actually simplifies the description of the network: now there are only documents connected to each other in a chord-like topology.

There are some consequences to this modification which we point out next.

1. The amount of work required for a physical node to participate in the network is bounded and controllable by the physical node. In particular, each document (aka virtual node) requires about $\log(m)$ work by the physical node when there are m published documents (and $\log^2 m$ work by the network). This is very useful when physical nodes might have wildly different connectivity abilities as is observed when comparing ethernet and modem connections.

2. The amount of work which any node can induce in the network is now bounded by $\log^2(m)$ per operation. In the standard chord protocol where there are n physical nodes and m documents, a physical node entering the network causes $O(\frac{m}{n} + \log^2 n)$ work. Since the set of documents and the set of physical nodes are unrelated $O(\frac{m}{n})$ could become arbitrarily large. Making the network do only minimal work for any operation implies that the network becomes more robust against denial of service attacks.

3. A physical node such as a web cache might have many (say k) documents. A straightforward implementation of the protocol would require $O(k \log^2 m)$ space to store the pointers for a set of nodes. However, a less straightfoward implementation can share some of these pointers, reducing space and network utilization. The particular means of sharing is more easily described in the context of the next modification.

4. One nice consequence of this modification is that any valid node in the network *is* a document. Consequently, issues with stale hashes never arise.

3 Distributed Nearest Neighbor

There is a basic difficulty associated with many peer-to-peer network designs: lack of locality. In particular, discovering the content in the network may require visiting nodes in Alaska, Australia, the US, Korea, and then Africa, even though the content we want is in our neighbors cache. This has several impacts:

1. High latency in searches.

2. Low bandwidth to the discovered sources of content.

3. Expensive network utilization. For many organizations, local network usage is essentially free while connections to the rest of the world are quite expensive.

One solution[1] is using a nearest neighbor datastructure where the distance between any two nodes is the round trip travel time. Under various semirealistic assumptions (see section 6), the minimum (over network topologies) maximum

[1] This solution was suggested elsewhere in broad form. The specific datastructure we suggest is new.

(over pairs) latency between k nodes must eventually be $O(\sqrt{k})$. In this setting, there is enough structure to allow a distributed "Cover tree" to calculate nearest neighbors. Then, instead of connecting a node to node 2^i in front of it, a node is connected to the nearest node in a hybrid metric of the form $\alpha_i d_{\text{hash}}(p,q) + (1-\alpha_i) d_{\text{ping}}(p,q)$. Here $d_{\text{hash}}(p,q) = |p-q|$ (thinking of nodes as hashes as integers) and $d_{\text{ping}}(p,q)$ is the latency for network traffic from p to q.

Again, there are some implications for this choice which we mention.

1. Hot spot removal: When a particular document is popular, it might be published and queried many times. This is fine for the nodes doing the publishing, but without localization, the hash just prior to the popular hash might become overloaded with network routing traffic. With localization, the definition of "just before" becomes "just before amongst all nearby (ping-distance) nodes". This distributed definition removes hot spots.

2. Adversarial robustness: In any protocol used by millions of people, we must ask ourselves how robust the protocol is against an adversary. The standard chord protocol is not very robust as an adversary can easily censor any particular hash, any particular query-maker, and generally corrupt the topology. In a localized version of the protocol, the adversaries power is limited by proximity. In particular, the protocol can be made so that adversaries with a large latency can not disrupt local (low-latency) use of the network.

3. Geometric robustness: The nearest neighbor algorithm works on any valid metric. By periodically renewing the datastructure, it even can be made to work on a changing metric.

4 A Chord Protocol for Caching (Chordcache)

4.1 The Steady State Topology

We first define the steady-state topology of the protocol, and then define the algorithms which achieve it.

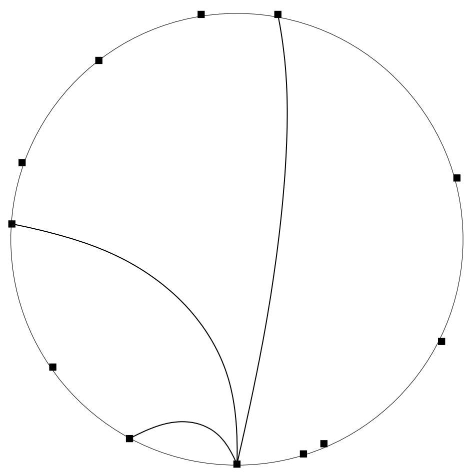
We first simplify our presentation by assuming that every node has only one piece of content. In general, a node on the network might of course have more than one piece of content, each treated as a logical node in the protocol. Naturally, this is optimizable by directly considering networks of nodes with many edges, but that is a worry for later.

For any node has a canonical name which is a 128 bit hash of it's content (for example 15030021b2ef408ad33fa62795fff1be) using some cryptographically strong hash function. We can regard this hash as a number in the field $GF_{2^{128}}$, implying that operations like $n_1 + n_2$ have a well defined meaning. Let's also suppose that any pair of nodes n_1, n_2 has some fixed measurable round-trip latency of $d(n_1, n_2)$. Note that we choose latency as our metric to optimize here, although sometimes bandwidth or other metrics may be important.

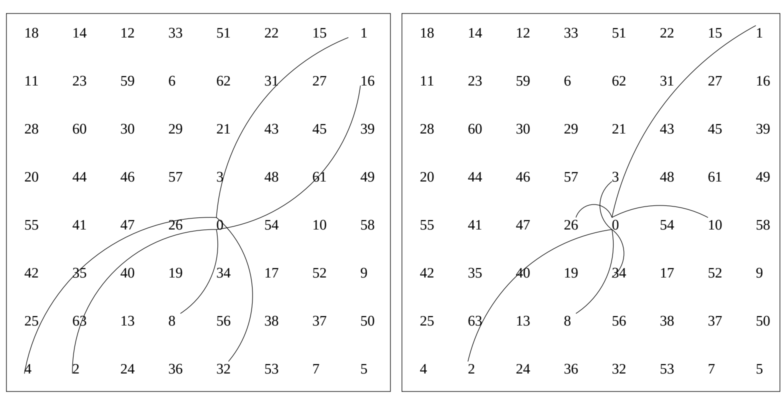
Each node has a set $e(n)$, of at most 128 directed edges associated with it. The ith edge $e(n)_i$ connects to a node n_i with minimal latency in an interval of size exponential in i.

$$n_i = e(n)_i = \arg \min_{n' \in (n+2^{i-1}, n+2^i]} d(n, n')$$

If the set $(n+2^{i-1}, n+2^i]$ is empty, then there is no edge. Since we are working over a field, we can place every node on a discretized ring with 2^{128} entries. The set of connections for each node is very similar to that in the chord protocol



This picture is very similar to what the chord protocol produces. The difference here is that we have explicitly chosen our chords in order to minimize latency. Another picture may be helpful. Suppose we have a large number of nodes arranged on a regular grid. What is the set of edges we use? Two picture, one for chord and one for chordcache are illustrative.



As designed, chordcache tends to have shorter edges than chord. (This permutation of node labels was chosen at random so it represents typical behavior.)

One point of difference between the chordcache protocol and the chord protocol is the treatment of duplicate content. Chordcache does not maintain the simple ring+chord structure when there are duplicate objects. To gain some insight into the behavior of chordcache imagine for the moment that there are two sets of nodes "nearby" and "far" nodes where all pairs of "nearby" nodes are closer than the distance between any "nearby" and "far" nodes. Then the subgraph of chordcache amongst the "nearby" nodes is a chord graph for the content which exists locally. Thus searches for locally duplicated content occur entirely within the "nearby" nodes.

We could imagine trying to run two versions of chord, one for the "nearby" nodes and one for all nodes. When searching, we could first search the "nearby" graph and then search the "all" graph if that failed. The chordcache graph accomplishes the same semantics in one unified search on one unified graph. It also does this without the sticky issue of defining and maintaining a "nearby" parameter.

4.2 Searching and it's Analysis

Any checksum can be located using binary halving of the 2^{128}-size search space. Furthermore, if no such checksum exists, this can be detected.

Search(checksum n^*, node n)

1. Let n_1, ..., n_I = the destination of n's edges.
2. If there exists $n_i \in e(n)$ with $n_i \leq n^*$
   (a) then Let $n' = \max\{n_i | n_i \in e(n) \text{ and } n_i \leq n^*\}$
   (b) if $n' = n^*$
      i. then return IP_address(n')
      ii. else Search(n^*, n')
3. else return Not_Found

This search algorithm can be easily altered to work on an interval rather than a set of points.

4.2.1 Analysis of Searching

Theorem: (recursion efficiency) If the node values are randomly distributed, the search algorithm recurses at most $O(\log m)$ times with high probability

Proof: (sketch) The "worst" case is when the metric $d(,)$ causes edges n_i and n_{i+1} to be "far apart". The farthest apart they can be is $\frac{n_{i+1}-n}{n_i-n} \leq \frac{2^{i+2}}{2^i} = 4$, implying that each recursion of "Search" eliminates $\frac{1}{4}$ of the search space. After $O(\log m)$ recursions occur, the search terminates with high probability.

Note: The base of the "logarithm" or equivalently the constant in the $O()$ could be improved by noting that the expected value of $\frac{n_{i+1}-n}{n_i-n}$ is less than 4.

Also note: the chord protocol has a slightly better logarithm.

We can state another theorem: If there are two sets of nodes N_1 and N_2 obeying $\forall p,q \in N_1 : \ d(p,q) \leq \epsilon$ and for all $p \in N_1$ and $q \in N_2$, $d(p,q) \geq 2\epsilon$, then any search for $p \in N_1$ originating in N_1 returns $p \in N_1$ (even if $p \in N_2$ also).

4.3 Dynamic Operations

Dynamic operations (adding and removing nodes) on the chordcache graph are *hard* in general. For example, suppose that there are two nodes n, n' on the moon which are right next to each other with $d(n,n') = 0.5 * 10^{-3}$ seconds. Each of these nodes is connected to the internet around the earth. Since speed of light delays to the earth are much greater than $0.5 * 10^{-3}$ seconds, the node n should have an edge to the node n'. Yet discovering n' by interacting with the chordcache network may be extremely difficult based only upon round trip latency distances.

Theorem: There exist very difficult choices of $d(n,n')$

Proof: By example

We can, however, find neighbors quickly using a modification of the "Cover Tree" datastructure. Let's encapsulate this possibly-hard-but-often-easy difficulty in an oracle nearest(n, I) which discovers the n' in the interval I which minimizes $d(n,n')$.

Note that nearest(n, I) *is* always tractable for chord and has essentially the same recursive complexity as search. Thus, $O(\log m)$ should be regarded as a lower bound on the complexity of nearest(n, I).

Also note that can construct nearest(n, I) to always find any existing node in the interval I with recursive step complexity $O(\log m)$ by using a variant of the above search procedure.

4.4 Node Addition

Node addition is relatively easy. Each edge of a node $e(n)_i$ is associated with a fixed interval $(n + 2^{i-1}, n + 2^i]$ implying that the following program fragment will discover the nodes:

1. for i in 1...128 do:
   (a) $e(n)_i \leftarrow$ neighbor$(n, (n + 2^{i-1}, n + 2^i])$

The previous program fragment is not quite enough to maintain the structure of the chordcache network because it may be that other nodes n' could have improved edges $e(n')_i = n$. It appears generally difficult to detect which edges might be improved, so one simple solution is to have every node periodically reinsert itself on a randomized interval of expected duration t. When n' reinserts itself into the network, it will discover n using the above code fragment.

4.5 Node Removal and it's Analysis

Nodes (and associated content) periodically leave the network. In the worst case, they may abruptly leave, so it is important that we use a protocol for node removal which does not rely upon the node being removed.

The fundamental problem with nodes leaving the network is that edges pointing towards the node effectively have distance ∞. The same background process of periodic reinsertion used for node addition fixes this problem. It is probably also helpful in practice to reinsert on demand whenever searches discover a broken edge.

5 Discussion

We can state several complexities based upon the parameters m = number of nodes, nearest= time complexity of finding the nearest neighbor and t, the periodic reinsertion rate.

Operation	Time complexity
Search	$O(\log m)$
Insertion	$O(\text{nearest} * \log m)$
Removal	0
Offline Work	$O\left(m\frac{\text{nearest}*\log m}{t}\right)$

The "offline work" entry is the amount of work that the entire network must do, so dividing by m results in the amount of work than any individual node does. When viewing this table, it's important to remember that nearest$\geq O(\log m)$.

6 The Physics of Locality

(This is just me ranting about why specifically two dimensions are to be expected. Perhaps it should be removed or pared down.)

Given that we have n active nodes in a network, what is the minimal addressing time between the n nodes? In the standard model, we assume that any node can address any other node in constant time. This assumption is a reasonable first-cut assumption for many analyses, but it's physically unrealistic difficulties imply that good performance with respect to this first-cut model does not translate into good performance in the real world.

To understand the basic physical difficulty, assume that a node converts some amount h of energy into heat per second. In any particular volume $V = \frac{4}{3}\pi r^3$enclosing n nodes, the heat energy hn must be dissipated through the surface area $S = 4\pi r^2$. We can regard the volume V (and it's n nodes) as some physical system which must dissipate hn heat per second in steady state.

There are several mechanisms for dissipating heat from a system.

1. Conductance: To use conductance, the system must be in contact with some material that conducts heat energy. Diamonds work very well, but many metals work reasonably well also. No known physical materials can conduct an arbitrarily large amount of heat through a fixed surface area, which implies that some area proportional to hn must be used to conduct heat away from the volume V.

2. Convection: To use convection, the volume V must be in contact with a gas or liquid. The gas or liquid is heated up and then expelled from the system via either natural or man-made processes. No known physical materials have a heat capacity (ability to absorb heat energy per unit mass) which is infinite implying that a finite amount of gas/liquid moved per second implies a finite amount of heat energy dissipated per second. Assuming finite rates for moving gas/liquid, the amount of gas/liquid that can moved into contact with (or through) V in a unit time is proportional to the surface are of V. Again, we see that V must have an area proportional to hn.

3. Radiation: Materials naturally radiate heat energy in the electromagnetic spectrum at a rate which grows as ST^4 where T is the temperature and S is the surface area of the body. Assuming the materials in V have a finite temperature, this implies that S must again grow as hn.

For each of the above methods, and every known material composition of V the number of nodes n generating heat h in the volume V is at most porportional to the surface area, $nh = kS$ where k is some (materials limited) constant. This argument does not imply that we can't have more than $\frac{kS}{h}$ nodes in the volume V—it just implies that we can't have more than $\frac{kS}{h}$ nodes active for long durations.

Assuming steady state, the above analysis implies that the number of active nodes within a distance r of some point is $\frac{k4\pi r^2}{h}$. Since communication is limited by the speed of light, this implies that only $\frac{k4\pi r^2}{hc^2} \propto r^2$ nodes can be communicated with in time $\frac{r}{c}$.

The principle objection to this analysis is that current systems are so sparse that the heat limit discussed above does not have any significant ramifications. This objection is technically correct since it is physically possible place the 10^8 (or so) nodes of the internet in a much smaller volume than Earth. However, we do not need to reach the physical limit before the scaling properties of the physical limit become important. Part of the reason why the nodes of the internet are spread out is because it is simply cheaper that way. Put another way, heat dissipation is a serious economic issue for hosting companies.

Another important reason to believe in the r^2 limitation is that (for now at least) the internet is confined to Earth. Since the number of points on the surface of a sphere within a (geodesic) distane r is proportional to r^2, we have another reason to believe that at most $O(r^2)$ nodes can be within communication time r of each other.

Another important objection to this analysis is that the communication mechanism may be so inefficient that these concerns are meaningless. This is a reasonable objection when considering modems, for example, but more modern mechanisms for communication appear to be nearly reaching the speed of light. For example, about 100 milliseconds of the latency in a 'ping' between Australia and the US is fundamental and unremovable. This is suprisingly close to observed latencies and dramatically longer than the latency between nodes on a common ethernet segment. By assuming that the communication time is directly proportional to the distance, we are essentially assuming that routing problem is "solved". This gross approximation appears to hold true over large distances.

Motivated by the above analysis, we can adopt a new model: For any set of n nodes, the communication time between a random pair of nodes is $O(\sqrt{n})$. In this new model, the search time to find content is $O(\sqrt{n}\log n)$. This is a rather severe penalty when a nearby much smaller set of nodes with size $k \ll n$ contains the desired data.

We might instead consider models where the communication time between a random pair of nodes in a set of size n is $O(n^\alpha)$ for $\alpha \geq \frac{1}{2}$. Indeed, if we fit the parameter α to the internet, we'll choose some $\alpha > \frac{1}{2}$. However, this model would only be an improvement for the internet as it exists today. In designing a protocol for tomorrow we might choose of model of $\alpha = \frac{1}{2}$ for the internet of tomorrow, where we (optimistically perhaps) hope that routing is solved problem in the future.

References

[1] The Chord project, http://www.pdos.lcs.mit.edu/chord/

[2] The Freenet Project, http://freenet.sourceforge.net/