Comparing LH* & Chord.
Private Communication May 2004

Witold Litwin

À:	istoica@EECS.Berkeley.EDU
Cc:	hari@csail.mit.edu; 'Frans Kaashoek'; 'Joe Hellerstein'; 'Ion Stoica'; 'Scott Shenker'; 'Sylvia Ratnasamy'; 'Richard Karp'; breitbar@cs.kent.edu
Objet:	RE: Chord & LH*

Hi Ion & al.,

I finally found the time to read the Chord paper (vacations helping). Sorry for the delay of this reply to your long enclosed message. Hope you got the short one saying I'll need more time. I read since your paper told at the site being submitted to the IEEE journal. If it is not out yet, I may suggest some corrections.

By now, I believe to have finally a knowledge by my own of your interesting work. Let me first sum it up to check with you. Or rather, more generally, - to present my basic understanding of key similarities/differences between Chord & LH*. Responding hopefully to some questions in your message, while postponing consequently some other detailed replies.

0. Both schemes address the problem of key search. Chord basically only as distributed lookup protocol. LH* as a data storage structure, apparently more classically.

1. Chord is basically for P2P with all nodes searching & storing data. LH* predated the P2P concept. It was intended for the client server work. The LH* client node only searches the data. The server node stores, forwards & sends IAMs (Image Adjustment messages) to the client upon the need. A node can however be client & server. It is then a peer as for Chord.

2. Chord and LH* necessarily map a physical node address into a persistent logical node address (ID...). A fundamental assumption of Chord is that it assigns the logical address as a persistent random integer in a large (2**m) interval. Chord maps then any key c into node C such that c =< C. A fundamental assumption of LH* is to assign to a new server node (peer) only, not to any client, the successive integer N >= 1. Here N is the number of server nodes with the logical addresses already assigned. The N value is tight-controlled as function of the storage load. For key to address mapping, bound to O...N-1 at any time, LH* uses a couple of hash functions: h(j-1,c) & h(j,c) where j scales with N.

3. For both schemes, the primary location of a key is the node where its address fits. So, some nodes have to split when the file scales. LH* triggers ordered splits to avoid the file overload. Keys move from node n (pointed by the split pointer) to new node N. Every bucket in any N-node file eventually splits. For Chord, a random split occurs when a new peer node P joins. Some keys from its successor move to P. A node may never split with non zero probability. Node load can perhaps vary more than for LH*. In both cases, the balancing that Chord calls "virtual servers" and that seems similar to that in Breitbart-Waikum Sigmod paper we spoke earlier, may be useful.

4. To speed up the key search with respect to the sequential search (hardly scalable) of the consistent hashing, Chord does its best to minimize the number of hops. For LH*, we talk about minimizing the number of forwards, but this is a synonym. Chord constructs the binary trees (tries) of its fingers at each node. LH* needs only at each node the level of hash function used for the last split. This is possible only because the node addresses are consecutive and deterministically assigned. The consequence on the key search are known. For Chord, the speed has to be O(log N). On the average about 1/2 (log N). E.g., 10 - 20 for a 1M-node file (P2P system). For an LH* client, it remains bound to at most two hops. On the average, it can be about zero in practice (two messages per search thus all together which is the absolute minimum). Besides, the search speed of a client improves with IAMs. I could not see such a notion for Chord. The stabilization alg. bears nevertheless some similarity in ideas.

Interestingly, what Chord does is probably the fastest for random logical node addresses. It is somehow equivalent of what do best known extensible hash schemes other than LH. Also, the address basically pointed to by i-th finger from node A that is A + 2**(i-1)+A, is the target of i-th split from LH* bucket A. Hence, it is also i-th successor of node A in your sense.

5. There is also a consequence on the unsuccessful key search. In Chord, it is crucial. The unsuccessful key search may silently fail (when a new node enters the picture). A silent failure of unsuccessful search does not seem yet generally accepted for a practical data structure. In LH*, the unsuccessful key search never fails. Also, the performance is even the same as for the successful search, i.e., about best ever possible. Notice that this is not always the case for a hash based org., e.g., the open addressing.

6. There are also basic implications on the addressing metadata size at a server node. Chord node needs the storage for the physical addresses of the fingers. LH* server needs the storage for N/2 to N addresses. This is much more. But remains easily practical, e.g., a couple of MB for a 1M-node file.

7. An LH* client can in contrast modulate at wish its metadata. Storing the max of the node addresses learnt by the client, i.e., up N perhaps, minimizes the average number of forwards. Alternatively, deciding to store less, even only one, increases this number towards only two however at worst, depending on N. A client can hook up to the system knowing only one address. It then improves the access performance incrementally with IAMs. For Chord the concept of a client does not seem elaborated at present. Chord new node can also hook up knowing only one existing address.

8. All Chord nodes perform same functions. Not in LH*. In addition to the distinction client - server, basically one node has also the actual file state data (n,i) and coordinates the split. It is typically node 0. This node may be replicated for reliability. There is also a version of LH* without the coordinator (the TODS paper). Like in Chord system. This version was not since worked out more. There was no apparent practical need yet.

9. Both orgs address the need for nodes randomly coming and leaving. LH* clients are those. Their arrival/leave does not bother any server's metadata. They could be the "free riders" in P2P, also frequent (50-70% of nodes according to Freenet papers). Chord has a harder time with those. Especially, the leaver has to move its keys to the successor & pointers among nodes should be adjusted. This is somehow equivalent to an LH* server leaving. That event is however considered as deterministically forced and usually less frequent. It basically results from the underload control. It triggers the merge operation and N := N -1.

This approach could be easily be adapted to the (new) P2P voluntary leave case. Besides, LH* servers may be protected against failures (unavailability) in various ways published since. Especially, by an erasure correcting coding in LH*rs (Sigmod 2000). In Chord, the node replication seems the main approach to this need at present.

Hope my above understanding of Chord correct. Please help correcting with your comments. It will help my perception of your innovative work and facilitate our further coop consequently. We can in particular discuss more if any of you comes for SIGMOD to Paris.

Best regards,

Witold Litwin <http://ceria.dauphine.fr/witold.html>
ACM Fellow <http://www.acm.org/awards/fellows_citations/litwin.html>

/Professeur U. Paris Dauphine/ <http://www.dauphine.fr/> /Directeur du CERIA <http://ceria.dauphine.fr/> / /Pl. du Mal. de Lattre, 75016 Paris, France/

Bureau A 410 (Nouvelle Aile)
tel & fax (0)1-44-05-48-80 ou tel 41-21 (secr.)

-----Message d'origine-----
De : istoica@EECS.Berkeley.EDU [mailto:istoica@EECS.Berkeley.EDU]
Envoyé : samedi 27 mars 2004 11:19
À : litwin.witold@wanadoo.fr
Cc : hari@csail.mit.edu; 'Frans Kaashoek'; 'Joe Hellerstein'; 'Ion Stoica'; 'Scott Shenker'; 'Sylvia Ratnasamy'; 'Richard Karp'; breitbar@cs.kent.edu Objet : Re:

Dear Witold,

Thank you for your e-mail. We certainly did a poor job of citing your work and discussing it in the context, and we apologize for this. We will definitely correct this issue.

After reading your ACM/TODS paper and going through your PowerPoint presentations (available on your home page) here is our understanding about the similarities and differences between DHTs and LH*. Please let us know if we are missing any important aspects.

At the high level DHTs and LH* are similar in that they both partition a linear ID space among a set of nodes.

However, DHTs and LH* differ in how they achieve and maintain this partition. In particular, they differ in the following three aspects:

1) The hash function in DHTs is used to map a document/key to an ID in the ID space. The hash function is independent of the size of the DHT and the ID space is fixed. DHTs use then a lookup/routing protocol to find the node that is responsible for that ID.

In contrast, the hash function in LH* is used to map a key directly to the ID of a server, where both the hash function and the size of the ID space depend on the system size. Then LH*, at least in its basic design, assumes that the client knows the mapping between the ID of each server in the system and its IP address, so the lookup operation (i.e., finding the machine with a given ID) is straightforward. (Even when a dynamic table is used to maintain the set of servers, as far as I understand from sec 3.5 of the ACM/TODS paper, in steady state, each client ends-up maintaining the addresses of all servers it has accessed data on.)

Thus, a DHT protocol is primarily concerned with mapping IDs to physical machines, that is, finding the physical machine responsible for a given ID; the mapping of keys to IDs is trivial. On the other hand, LH* is primarily concerned with mapping a key to a logical ID; finding the machine with a given ID is arguably the secondary task.

2) The way in which the system evolves is different. In LH* the system grows/shrinks as the load increases/decreases. The system starts with one node/server and then grows when the node becomes "overloaded". In contrast, DHTs grows/shrinks as nodes join and leave the system, and data is spread across all the available nodes.

3) DHTs were designed for highly dynamic environments where node failures are common. This assumption lead to designs in which each node knows only about a limited number of other nodes in the system (e.g., O(log N)). Thus, any change in the system such as a node joining or leaving affects only a relatively small number of other nodes.

In addition, DHTs protocols are *symmetric* in that every node plays the same role in the topology, i.e., every node is a root of an embedded tree.

Finally, because DHTs were designed from the beginning for the Internet, minimizing the lookup latency is an integral part of the system design. Virtually all DHTs use proximity aware routing algorithms, and various replication/caching strategies that allow a requester to find a nearby replica of the requested data.

In contrast, it seems to us that LH* was designed for a more stable environment, where the node (server) population is quite stable over time. In such an environment it makes sense to have a coordinator that knows about all the servers in the system. Again, in the original DHT designs there is no entity that knows about all nodes in the system.

Also, it appears to us that in LH* the server roles are not completly symmetric, as a server maintains state about all the buckets it has created, so servers which joined earlier will maintain more state than servers that joined later. Also, as I mentioned above, it seems each client will end-up maintaining the addresses of all servers on which it has accessed data. Actually, this is required to achieve one hop lookup operations in the common case.

In conclusion, it is true that LH* and DHTs offer the same abstraction, but the way they achieve it is quite different.
This difference is mainly the result of the different assumptions that each system makes about the dynamicity and the reliability of the underlying network system. Please let us know if we are missing something.

Best regards,

Ion

-------- Original Message --------
Date: Tue, 23 Mar 2004 00:16:02 +0100
From: Witold Litwin <litwin.witold@wanadoo.fr>
Reply-To: litwin.witold@wanadoo.fr
To: istoica@cs.berkeley.edu
CC: yuri Breitbart <breitbar@cs.kent.edu>, karp@cs.berkeley.edu, Joe Hellerstein <jmh@cs.berkeley.edu>, kaashoek@lcs.mit.edu

Dear Ion,

I have attended last week the EDBT conf. At Springer's booth, your book from P2P II was on display. I bought a copy. So your kind offer from some months ago, to send a copy when published, does not need to be fulfilled anymore. Thanks. The book is also very instructive and nicely edited. Congratulations to both editors.

At EDBT I gave an invited talk on SDDSs and P2P computing. After the talk I got a good question, I guess. To which I did not have a good answer. Even after looking into your book later on. The question was about the relationship of LH* (Scalable Distributed Linear Hashing)to DHTs for structured P2P systems, as you called them, in Choord it seems. Apparently, the question was triggered by the lack of references in the Choord papers to that work, and to the SDDS papers in general. I did not look through the Choord papers. I am yet on the plane on the way back from EDBT. So I do not have a personal opinion. If it is true, however, then perhaps you might help and I'll forward the reply. It is true in contrast for most of Choord derived and related papers in the workshop. I could look through them on the previous segment of my flight.

The story is somehow intriguing. On the one hand, main papers on SDDSs, including on LH*, have been published in ACM-SIGMOD, VLDB, EDBT, ICDE and ACM-TODS. These are the most respectable database research conf/journals. Hence not exactly those where a paper is supposed to get unnoticed. Besides, the 1st papers appeared at Sigmod already 1993. Hence, one had ample time to notice them. As many citations to it (including D. Knuth's new edition) show. Perhaps it is worth to suggest to P2P folks to talk more to Joe (Hellerstein), very aware of the SDDS work (or Hector, or S. Gribble alternatively). All these folks are able to suggest the database conferences journals etc worth reading in the P2P context.

There is little more on to the mystery, given you're at UCB. The LH* algorithm was proposed while I was visiting at UCB for two years. The 1st public talk was on the kind invitation of R. Karp to his research group seminar (reason while I cc him here). Back to 1992-93. The SDDS work was rather well known there at those old times (when the term P2P did not even exist yet).

A technical consequence of all this is that some claims or results in the seminar's papers are to my understanding at least partly unfounded. The number of forwards between the data servers (or as one now calls it hops in some papers in your book) to deliver a message to the correct server among N is 2 at worst for LH* (which makes 4 messages all together per search in the worst case, and 2 usually). Hence the claims in some papers that log2 N is the lowest bound, given the fingering DHT scheme in Choord or close to it in Koord etc., seem wrong. These results also ignore the papers by Nardelli & al, and Kroll & Widmayer, also a Sigmod paper BTW. I attend in two weeks the ICDE in Boston, if Mr. Kashoek is around we may have an instructive discussion. The concept of a virtual server for load balancing, is quite close to the approach by Breitbart & Waikum in their Sigmod paper, from 93 or 94 I believe. Not quoted neither by the papers in the seminar I could look through. I cc this message to Prof. Breitbart for a direct discussion if anyone is interesting. The proposals on fault tolerence for P2P, the high-availability especially, ignores our Sigmod 2000 paper on RS erasure correcting coding for this purpose in general in the scalable distributed environment, P2P & Grids included. Etc.

Coming back to the original question from my audience, would the lack of the discussed referencing to earlier related work indicated a gap between the emerging P2P community and the database community ? If so, we should correct the anomaly. But perhaps there is another good answer. May be, I am just speculating, it is due to a different terminology, as terms P2P or Grid did not exist by those times. Folks rather spoke about multicomputers in Tanenbaum sense, networks of workstations, client nodes and/or server nodes (while a P2P node seems systematically both) etc. May be, it is due to other things. So, I am in the waiting loop for your expert opinion.

Best regards,

Witold Litwin <http://ceria.dauphine.fr/witold.html>

/Professeur U. Paris 9 Dauphine/ <http://www.dauphine.fr/>
/Directeur du CERIA <http://ceria.dauphine.fr/> /
/Pl. du Mal. de Lattre, 75016 Paris, France/

Bureau A 410 (Nouvelle Aile)
tel & fax (0)1-44-05-48-80 ou tel 41-21 (secr.)