ISSN : 0976-8491 (Online) | ISSN : 2229-4333 (Print)

Analysis in Heterogeneous Distributed Hash Tables

[1]CVVN Varun, [2]A Siddhartha Reddy, [3]V. Siva Nagi Reddy, [4]G.B.V.Kishore, [5]S. Chinna Gopi

[1,2,3,4,5]Dept. of CSE, KL University, Vijayawada, AP, India

Abstract

We present a scheme for evenly partitioning the key space in distributed hash tables among the participating nodes. The scheme is based on the multiple random choices paradigm and handles both node joins and leaves. It achieves, with high probability, a ratio of at most 4 between the loads of the most and least burdened nodes, in the face or arbitrary node arrivals and departures. Each join or leave operation incurs message cost that is, with high probability, O ($\log^2 n$), where n is the number of nodes, and causes the re location of keys from at most one node (for joins) or three nodes (for leaves).In this paper, A version of heterogeneous systems, where the capacities of nodes to serve keys can vary widely.

Keywords

Peer-to-Peer, Distributed Hash Tables, Load Balancing, Heterogeneous Systems

I. Introduction

The last several years have seen the emergence of a class of structured peer-to-peer systems that provide Distributed Hash Table (DHT) abstraction [1-4]. In such structured systems, a unique identifier is associated with each data item and each node in the system. The identifier space is partitioned among the nodes that form the Peer-to-Peer (P2P) system, and each node is responsible for storing all the items that are mapped to an identifier in its portion of the space. Distributed Hash Tables (DHTs) [1,5,9,11-14,16-17], are data structures used to organize highly dynamic, massive, decentralized distributed systems, such as peer-to-peer networks. A DHT maps keys (i.e., resource identifiers) into a hash space, which is partitioned among the nodes in the network. Each node is responsible for (the resources whose keys are hashed into) its segment of the hash space. When a node joins the system, the partition of the hash space is perturbed slightly, so that the new node can be given its own segment of the hash space, pieces of which previously belonged to a small number of old nodes. Similarly, when a node leaves the system, its segment is redistributed among a small number of the remaining nodes. In addition to the join and leave operations, a DHT supports the lookup operation which locates the node responsible for (the resources identified by) a given key. An important goal in the design of DHTs is to achieve a balanced partition of the hash space among the nodes in the system. It is often desirable that each node assumes responsibility for a portion of the hash space that is proportional to its power (measured in terms of its processor speed, available bandwidth, and/or storage capacity), and that this property is maintained as nodes join and leave the system.

This is because, typically, the number of keys a node is responsible for serving and the amount of routing traffic the node handles are proportional to the size of the segment that the node is associated with. We quantify the notion of load balance in DHTs in the following natural way. Assume each node has a positive weight, that is proportional to its power, i.e., for any nodes n1, n2 of weights respectively w1, w2, with $w1 \geq w2$, n1 can manage a segment of the hash space that is w1/w2 times larger than that n2 can handle. The weighted size of a node's segment is the quotient of the segment's size divided by the node's weight. Finally, the balance ρ of a DHT structure is the ratio of the maximum weighted size of any node segment to the minimum weighted size. In this paper, we describe a simple and intuitive protocol for managing the partition of a DHT's hash space into node segments, as nodes join and leave the system.

II. DHT Structure

In most early DHT structures [9,11,13], each node (upon arrival) chooses at random a point in the hash space (typically, the unit interval (0, 1), and becomes associated with the points of the hash space closest to the selected point (with respect to some distance function). Assuming random node departures, this scheme guarantees that the ratio of largest to average node segment size is $\Theta(\log n)$, with high probability [6, 17]. However, we can show that the ratio of average to smallest segment size is Ω (n) with constant probability. Therefore, assuming homogeneous nodes, ρ is Ω (n log n), with constant probability malicious attacks on the network difficult. In another early approach [14], a new node splits in half the node segment containing a randomly selected point, and assumes responsibility of one half. In the pure join model, the balance of this scheme is better than the previous one, but we can show that ρ is still Ω ($\log^2$ n/ log log n), with high probability. (The ratio of largest to average segment size is the same as before, while the ratio of average to smallest is Ω(log n/ log log n), with high probability.) In both this and the previous approaches, if every physical node acts as Ω(log n) virtual nodes, each associated with a distinct segment, then constant ρ is achieved, with high probability [6, 17].

However, this approach amplifies by a factor of Ω (log n) the numbers of links a node should maintain, and (consequently) increases the complexity of join and leave operations. Five recently proposed load balancing schemes guarantee constant ρ, with high probability, for homogeneous nodes, while assigning a single segment per node. Abraham et al. [1] and Naor and Wieder [13] independently suggested and analyzed the join scheme our (unweighted) protocol uses. However, neither work provides a method for a departure that provably achieves constant ρ. Adler et al. [2], propose and analyze a scheme for joins in a hyper cubic overlay structure.

Roughly, a new node splits in half the longest segment among the ones associated with the node responsible for a randomly selected point, and its Θ(log n) neighbors in the overlay structure. They also describe a procedure for leaves, but they do not show it maintains constant ρ, although experimental results suggest that it may do so. The message and (parallel) time complexity of node joins are Θ (Lm(n) + log n) and Lt(n) + O(1), with high probability, respectively, and the number of nodes affected is constant. The next two approaches are not tied to specific overlay structures, and provably guarantee constant ρ in the face of both joins and leaves. They [7], present a scheme with message and time complexity respectively Θ(log n · Lm(n)) and Lt(n) +O(log n), with high probability. The number of nodes affected by a single operation is O (log n), with high probability, and Θ (log log n), on expectation. we [10], proposes a very efficient scheme where operations are, roughly, performed as in our protocol, but instead of random segments the scheme examines a logarithmic number of consecutive segments in the neighborhood of a random point. The message and time complexity are respectively Lm (n) + Θ

(logn) and Lt (n) + Θ (logn), with high probability, and constant number of node segments are modified per operation.
Finally, in a very recent paper they [8], present a load balancing scheme that combines two different approaches: the multiple random choices scheme, and the "neighborhood search scheme" of [10]. Their protocol is overlay independent, and provably guarantees constant ρ only in the pure join model. The complexity of joins depends on two system parameters r and v, with rv = Θ(logn), where r represents the number of random points selected, and v represents the number of (consecutive) segments examined in the neighborhood of each random point. The message and time complexity per operation are respectively rLm (n) + rv and Lt (n) + v, with high probability. All the above approaches assume homogeneous nodes. In particular, a balancing scheme that assigns a single (contiguous) segment per node would result in significant reduction in the number of links in the network compared to the approach that uses virtual nodes. For instance, in a system where half of the nodes are twice as powerful as the other half, a scheme that assigns a single contiguous segment per node would result in the more powerful nodes having (roughly) half the number of outgoing links and the same number of incoming links as in the virtual nodes approach, while the less powerful nodes would have the same number of outgoing and one third fewer incoming links.

III. Protocols for Balanced Partitioning

A. A Note on the Model

The load balancing protocols we present in the next two sections are general enough to be used in any DHT design whose hash space is a line segment or a ring. For concreteness we will assume that the hash space is the unit interval I = [0, 1). For efficiency, we will also assume that every node maintains links to its immediate successor and predecessor nodes in the hash space. As is typical in the analysis of load balancing algorithms for DHTs, we only consider the case where node joins and leaves occur sequentially. We expect the protocols should perform well up to some degree of concurrency, as well.

B. S & M Protocol

This scheme assumes that all nodes are equally powerful, and aims to partition the hash space evenly among them. Node joins and leaves are performed in such a way that, at any time, the length of the segment associated with each node is an integer power of 1/2 (potentially a different one for different nodes), and its endpoints are integer multiples of its length. We call such a segment of length $1/2^d$, for some d € N, a d-segment; we also say that this segment has depth d. Finally, we define the sibling of a d-segment λ, denoted., to be the unique d-segment such that λ U. is a(d − 1)-segment.

C. Protocol Description

To join the system, a new node n, first requests the depth of the segment associated with a node n0 that is known to be in the system already. Let d be that depth. Then, n issues (via n0) lookup requests for each of [α+d+β+] keys selected at random, where α+ and β+ are positive system-wide parameters. Let n be a node associated with a longest segment, among those returned by the lookup requests. Then, the segment of n is split into two halves, and each of n, n is associated with one half. The leave procedure is, in a sense, symmetric to the join process. Suppose some node n, who is currently in the system and is associated with a d-segment λ, d > 0, wishes to depart. Before n does do, it looks up the nodes that are responsible for [α−(d + 1)+ β−] keys selected at random, where, α−, β− are again positive system parameters. Let n| be a node associated with a shortest segment among the looked up nodes, and let λ| denote that segment. We have two cases, depending on the length |λ|| of λ.
|λ|| < |λ|: If λ' is associated with a single node, say (i.e., λ' is not currently split), then λ'and λ' are merged and are associated with the resulting segment and λ, respectively, and n leaves the system. If λ' is not associated with a single node (i.e., λ' is currently split), then n sequentially the nodes associated with sub segments of λ' , until it finds a pair of nodes n1, n2 associated with two sibling segments. (Note that we can always find such a pair of nodes.) Then, the two sibling segments are merged, n1 and n2 become associated with the newly created segment and λ, respectively, and n departs from the system.
|λ' |≥|λ|: The procedure is almost identical to that described in case (a) if we consider n in place of n' . So, we first attempt to merge λ with λ', and if this is not possible (because $\bar{\lambda}$ is split) we merge two sibling node segments that are sub segments of $\bar{\lambda}$.

D. Protocol Properties

The above protocol provides strong load balancing guarantees. Roughly speaking, if we start from a sufficiently balanced initial state, then at all times during an arbitrary sequence of k joins and/ or split operations, $\rho \leq 4$, with probability 1 − O (k/Nb), where N is the minimum number of nodes in the system during this sequence, and b > 0 a constant that can be made arbitrarily large by choosing large enough parameters α^+, β^+, α−, β− (independently of k or N). A more formal statement of this property and an outline of its proof are presented in Section 3. We note parenthetically that the protocol is "self-correcting" in the sense that starting from an arbitrary (valid) state, a sufficiently balanced state is reached after a large enough number of steps.
It is easy to see that the message complexity of a join operation in an n-node system with constant ρ is O (log n · Lm (n)). This is not optimal since other approaches require as little as O (log n) + Lm (n) messages [10]. However, if the probing lookups are executed in parallel (instead of sequentially), the time complexity of both join and leave procedures reduces to O (1) + Lt (n). Finally, a node join affects only the segment associated with a single node, and a node leave affects the segments associated with at most two nodes (in addition to the departing node). Thus, for constant ρ both operations result in relocating keys that fall in an O(1/n) fraction of the hash space, which is optimal.

E. Weighted S & M Protocol

The scheme we present in this section is an extension of the protocol described in Section III, that takes into account the relative power of nodes. In particular, it presumes that each node n has a weight, denoted w(n), that is an integer power of 2 between 1 and some system-wide parameter W (also a power of 2), and tries to partition the hash space in such a way that each node is associated with a segment of length proportional to its weight. Roughly speaking, this scheme arranges nodes into (virtual) groups, and uses the same technique as the un weighted S&M Protocol to achieve for each group what that protocol achieved for single nodes. Below we describe the protocol in more detail.
The hash space is partitioned into segments, one for each node, such that at any time there is a (unique) many-to one mapping of the set of nodes to a set of groups with the following properties: (i) the union of the segments of all nodes in a group, called a group

segment, is a d-segment, for some $d \geq 0$ (potentially, a different d for different groups);and (ii) the sum of the weights of all nodes in a group, called the group's weight, is between W and $2W - 1.2$ We denote the weight of a group G by w(G). The details of how a group segment is partitioned among the nodes of the group are specified by a part of the balance algorithm we call the group management protocol. In addition, the group management protocol describes how to perform the group operations listed below. Note that, these operations may only modify segments associated with nodes of the groups mentioned in each operation.

T1: Add node n to group G. If $w(G) + w(n) \geq 2W$, then G is split into two groups, each associated with one half of s segment.

T2: Given two groups G, G' associated with sibling segments, and a node n in G, remove n from the system. If $w(G) + w() - sw(n) < 2W$, then G and are merged into a single group.

T3: Given a group G, a node n in G, and two other groups G1, G2 associated with sibling segments, remove n from the system in such a way that either (i) the value of w (G) after the operation is greater or equal to that before, or (ii) G is split into two groups. Moreover, G1 and G2 may be merged into a single group.

The group management protocol can be designed independently of the rest of the load balancing algorithm, as long as it adheres to the above specification. In the next two paragraphs, we first describe the Weighted S&M Protocol in detail assuming an arbitrary group management protocol, and then we discuss specific group management protocol designs.

IV. S&M-Processes

We define two families of discrete stochastic processes that describe models for adding and removing segments in binary partitions. The first family, called actual S&M-processes, simply formulates as a stochastic process the S&M Protocol described in Section 2.2. The second family, called virtual S&M-processes, describes a slightly different, less complicated model. We introduce virtual S&M-processes because they are easier to analyze, and their analysis yields results that apply to actual S&M-processes, as well.

The sample space of an actual or virtual S&M-process is a set of finite sequences of binary partitions or sorted binary partitions, respectively. Both types of processes are parameterized by an initial partition, four positive constants called splitting factor/ term and merging factor /term, and a finite binary sequence of +/−'s, called event list. Actual S&M processes have an additional parameter, called list of points, which is a sequence of points from I of the same length as the event list.

V. Analysis of Weighted S&M Protocol

The analysis of the weighted version of the protocol is similar to that of the unweighted version. Again we use an aggregated representation of the system state, and study the transitions in this simplified (virtual) state space. This time, however, the abstraction we use is slightly more involved than the one we used in the analysis of the unweighted protocol (i.e., sorted binary partitions and virtual S&M-process), since here we need to also take node weights into account. More precisely, the information we maintain for each system state is the distribution of the group segment lengths, the distribution of group weights for groups associated with longest group segments, when the total length of these segments is below some threshold, and the distribution of group weights for groups associated with shortest group segments, when the total length of these segments is below a threshold.

As in the analysis of the unweighted protocol, we show that in states with a large number of longest group segments joins add nodes to longest group segments almost with certainty (for large enough system parameters). We also bound the number of joins required until all longest group segments are split when starting from states with a small number of longest group segments. The additional complication introduced by the fact that not all joins result in group splits is overcome by assuming "adversarial" weight selection, i.e., we assume nodes added in longest group segments have weight 1, while nodes added in other groups have weight W. Thus, a longest group segment must be selected W times before it is split, while a join in a non longest group segment always causes a group split. Analogous statements are shown to hold for node leaves. Finally, we combine these results. This analysis shows that the Weighted S&M Protocol supports for the group segments the same load balancing guarantees the un weighted protocol provides about node segments.

References

[1] I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, E. Pavlov", A generic scheme for building overlay networks in adversarial scenarios", In Proc. International Parallel and Distributed Processing Symposium (IPDPS 2003), pp. 40.2, April 2003.

[2] M. Adler, E. Halperin, R. Karp, V. Vazirani,"A stochastic process on the hypercube with applications to peer-to-peer networks", In Proc. 35th Annual ACM Symposium on Theory of Computing (STOC 2003), pp. 575–584, June 2003.

[3] Y. Azar, A. Border, A. Karlin, E. Upfal,"Balanced allocations", SIAM Journal on Computing, 29(1), pp. 180–200, 1999.

[4] P. Fraigniaud, P. Gauron,"The content-addressable network D2B", Technical Report 1349, RI, University of Paris-Sud, France, 2003.

[5] K. Hildrum, J. Kubiatowicz, S. Rao, B. Zhao,"Distributed object location in a dynamic network", In Proc. 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA 2002), pp. 41–52, August, 2002.

[6] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, R. Panigrahy,"Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web", In Proc.29th Annual ACM Symposium on Theory of Computing (STOC 1997), pp. 654–663, May 1997.

[7] D. Karger, M. Ruhl,"Simple efficient load balancing algorithms for peer-to-peer systems", In Proceed. 16th ACM Symposium on Parallel Algorithms and Architectures (SPAA 2004), pp. 36–43, 2004.

[8] K. Kenthapadi, G. Manku,"Decentralized algorithms using both local and random probes for P2P load balancing", In Proc. 17th ACM Symposium on Parallel Algorithms and Architectures (SPAA '05), July 2005. (to appear).

[9] D. Malkhi, M. Naor, D. Ratajczak,"Viceroy: a scalable and dynamic emulation of the butterfly", In Proc. 21st Annual Symposium on Principles of Distributed Computing (PODC 2002), pp. 183–192, July 2002.

[10] G. Manku,"Balanced binary trees for ID management and load balance in distributed hash tables", In Proc. 23rd Annual Symposium on Principles of Distributed Computing (PODC 2004), pp. 197–205, July 2004.

[11] G. Manku, M. Bawa, P. Raghavan,"Symphony: Distributed hashing in a small world", In Proc. 4th USENIX Symposium on Internet Technologies and Systems (USITS '03), pp. 127–140, Mar. 2003.

[12] P. Maymounkov, D. Mazieres. Kademlia,"A peer-to-peer information system based on the XOR metric", In Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), pp. 53–65, March 2002.
[13] M. Naor, U. Wiede,"Novel architectures for P2P applications: the continuous-discrete approach", In Proc. 15th ACM Symposium on Parallel Algorithms and Architectures (SPAA '03), pp. 50–59, June 2003.
[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker,"A scalable content-addressable network", In Proc. ACM SIGCOMM 2001 Conference (SIGCOMM 2001), pp. 161–172, August 2001.
[15] A.Richa, M.Mitzenmacher, S. Sitaraman,"The power of two random choices: A survey of techniques and results", Sept. 2000.
[16] A. Rowstron, P. Druschel,"Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems", In Proc. 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), pp. 329–350, November. 2001.
[17] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan,"Chord: A scalable peer-to-peer lookup service for internet applications", In Proc. ACM SIGCOMM 2001 Conference (SIGCOMM 2001), pp. 149–160, August 2001.

Chamarthi Veera Venkata Naga Varun received his B.Tech in Information & Technnology and Engineering from Nalanda Engineering College, Andhra Pradesh, India, in 2009. He is Pursuing M.Tech in Computer Science and Engineering in KL University, A.P, India during 2010-2012. His research invites Data Mining and Knowledge Discovery, Data Warehousing and Database System.

Aregakuti Siddhartha Reddy received his B.Tech in Computer Science and Engineering from Nalanda Engineering College, Andhra Pradesh, India, in 2010.He is Pursuing M.Tech in Computer Science and Engineering in KL University, A.P, India during 2010-2012. His research invites Data Mining and Knowledge Discovery, Data Warehousing and Database System.

G. Bala Venkata Kishore received his M.C.A degree from Andhra Loyola College, Vijayawada, Andhra Pradesh, India; He is Pursuing M.Tech in Computer Science and Engineering in K.L. University, A.P, and India during 2010-2012.