Towards Balanced Allocations for DHTs

George Tsatsanifos[1] and Vasilis Samoladas[2]

[1] National Technical University of Athens
gtsat@dblab.ece.ntua.gr
[2] Technical University of Crete
vsam@softnet.ece.tuc.gr

Abstract. We consider the problem of load-balancing structured peer-to-peer networks. Load-balancing is of major significance for large-scale decentralized networks in terms of enhanced scalability and performance. Our methods focus mainly on task-skew. Specifically, we address the problem with general rigorous algorithms on the basis of migration. In particular, the cornerstones of our methods are the notions of *virtual nodes*, *replication* and *multiple realities*. Finally, our work is complemented with extensive experiments.

1 Introduction

We revisit the problem of load-balancing structured p2p networks [8]. Our intention is to develop a realistic balancing paradigm, which can be used on top of any DHT overlay, mitigating load imbalance effects, in order to enhance performance and therewith scalability. Our methods are focused mainly on task-skew. We address the problem with general algorithms based on migration, reducing the problem to a balls-and-bins game by facing hosts as bins and virtual nodes as balls. More specifically, we study rudimentary balancing techniques, namely *virtual nodes*, *replication*, and *multiple realities*, over specific performance evaluation metrics. By scrutinizing each method in isolation, we draw valuable conclusions on how they can interoperate with each other, and augment them by applying each a balls-in-bins model.

The remainder of this paper is organized as follows. Section 2 reviews relevant literature. Section 3 discusses augmented rudimentary balancing techniques. Section 4 evaluates our work, and Section 5 concludes.

2 Related Work

The common balancing paradigm for DHTs consists of randomized hashed functions that are supposed to distribute data load among peers in a uniform fashion. However, this approach is far inadequate, especially in cases where certain parts of key space receive disproportional portions of popularity. Specifically, if node and item identifiers are randomly chosen, there is a $\Theta(\log n)$ imbalance factor in the number of items stored at a node. Mercury [2] supports explicit load balancing using random sampling. *The threshold algorithm* [4] consists of a load-balancing protocol whereby each tuple-insert or delete is followed by an execution of the load-balancing algorithm, which may involve moving sequential data across peers. Their rationale that a node attempts to shed

A. Hameurlain et al. (Eds.): DEXA 2011, Part II, LNCS 6861, pp. 485–492, 2011.

its load whenever it increases by a factor δ, and attempts to gain load when it drops by the same factor. Nevertheless, the threshold algorithm cannot be applied to systems indexing tuples along many dimensions. Karger and Rhul propose a family of caching protocols to decrease the occurrence of hotspots. *Address-space balancing* [6] improves consistent hashing in that every node is responsible for a fraction of the address space with high probability. The protocol is dynamic, with an insertion or deletion causing other nodes to change their positions. On the other hand, *Item balancing* [6] targets at the distribution of items to nodes. Rao et al. in [7] introduce the notion of the *virtual server*, a single node of the underlying DHT contrary to the physical host being responsible for more than one virtual servers. Most importantly, the topology of the virtual servers should be able to adapt to dynamic changes of the actual network accordingly.

3 Balanced Allocations

In this section, we augment existing rudimentary balancing techniques with algorithms that enhance performance. More specifically, we reduce the problem to a balls-in-bins game by facing peers as bins and virtual nodes as balls. We now delineate some terminology conventions we have made for distinguishing certain notions we use. We define as a *node* the entity responsible for some part of the key space of the overlay, which was assigned according to the overlay protocols. As far as this work is concerned, nodes and peers are two distinct notions. Henceforth, *peers* serve as node containers. A peer's task is to deliver incoming messages to the appropriate comprised nodes and forward all their outgoing messages to their destination.

Most of the prior works in balancing use a straightforward approach for random and arbitrary assignment of balls to bins, requiring no knowledge about balls, bins, their structure and their loads. On the other hand, our methods are iterative allocation schemes based exclusively on local knowledge that exploit balls-in-bins games with minimum makespan scheduling approximate solutions [1]. To elaborate, assume that there has already been made some sort of assignment of nodes to peers. Balancing takes place among the nodes of a neighborhood, where a peer's neighborhood corresponds to the union peer-set of all peers containing nodes from the routing tables of the peer's comprised nodes. For each iteration a random peer from an unbalanced peer-set is picked, and all nodes from linked peers are reassigned successively to the bins of that specific peer-set, starting from the heaviest node and assigning it to the lightest peer of the peer-set. In essence, our methods constitute infinite processes, as they are repeated sequentially, and we stop when no significant changes take place in the structure and a convergence criterion has been met.

The principle of the *multiple realities* technique is to maintain multiple, independent coordinate spaces, with each node in the system being assigned a different zone in each coordinate space. Data tuples are being replicated in every reality, enhancing this way, data availability and fault-tolerance. We consider the case where n peers participate in the network in all g realities. We also impose a limitation for a peer to participate in each reality with one node only. In order to balance, we reassign the nodes from all realities of a selected peer's nodes' vicinities, in such a way that all peers acquire nodes with approximately the same summing loads. When a peer "broadcasts" its request to

all realities simultaneously, the fastest answer is returned to the user, and thus latency ameliorates. Thus, when a lookup query is enacted in all g realities simultaneously requires g times more messages to be resolved.

For the *parallel universes* technique we aim at reducing latency by invoking queries simultaneously in all realities and get the fastest answers. Since balancing is our main concern, we force peers to enact their queries only in one of the realities, selected randomly and independently. Nonetheless, data insertions and deletions apply to all realities; otherwise the system is inconsistent, in that identical requests in different realities would yield different results. However, no other special consideration has been made, other than creating and maintaining redundant overlays. Hence, this comes at a cost of a fixed replication factor. In effect, along with overweight areas of space, the underweight ones are replicated as well. Our *local allocation* method assigns the nodes of the currently heaviest peer and its associated peers to the lightest peer among that peer-set, with respect to their summing load in all previous realities. The redundancy introduced in both schemes is fixed and equal to the number of realities g, and we expect it is kept in $O(\log n)$. However, message overhead is rendered obsolete.

Algorithm 1. Pseudo-Inflationary balancing algorithm

```
 1: repeat
 2:     state = State()
 3:     h = heaviest()
 4:     for u in h.universes do
 5:         hosts = minHeap()
 6:         nodes = maxHeap()
 7:         for j in h.nodes do
 8:             nodes.push((l_j, j))
 9:         end for
10:         hosts.push((L_i^{<u}, i))
11:         cache(state, h)
12:         while not nodes.empty() do
13:             (L_i, i) = hosts.pop()
14:             (l_j, j) = nodes.pop()
15:             assign(i, j, u)
16:         end while
17:     end for
18: until allocationChange(state)
```

The purpose of the *virtual nodes* technique is the even allocation of keys to nodes, by associating keys with virtual nodes, and mapping multiple virtual nodes (with unrelated identifiers) to each peer. This method results in assigning many nodes to a single peer. Intuitively, this will provide a more uniform coverage of the identifier space. For example, if we allocate $\log n$ randomly chosen virtual nodes to each peer, then each of the n bins will contain $O(\log n)$ nodes, with high probability. This does not affect the worst-case path length. On the other hand, the maintenance cost for a peer congregating many nodes (eg., updating routing tables) increases readily. The virtual nodes technique can be applied to any load function and type of skewness, enhancing this way flexibility and functionality. Besides, what load consists of is problem specific. In

principle, n hosts allocate the $m = g \times n$ nodes of the overlay, where $g > 1$. Hence, the maximum load of the busiest peer equals to the sum of the loads of its comprised nodes, and thus, can never be less than the load of the busiest node. This method is unsuitable for very large networks, since the fact that max process throughput $\Lambda_{\max}$ has a concave behavior. As a result, it would diminish instead of improve after some specific value of the overlay size. Therefore, given an overlay of specific size, there are certain values of g that ameliorate the maximum throughput. Thus, this technique provides a network of size n with the throughput of an overlay of size m, a property that can be graphically interpreted as a transposition in the $\Lambda_{\max}$ graph, as messages are routed throughout the larger overlay. More importantly, the virtual nodes technique is devoid of redundancy. A peer may contain many nodes but each node can be hosted in one peer exclusively (many-to-one scheme). Our *local allocation* scheme uses a peer's limited knowledge. We select a peer i that invokes a balancing procedure among all peers in i's routing table. At first, all peers participating in the process deploy their nodes. Then, we convey all nodes one-to-one and we assign the heaviest node to the lightest peer. In particular, we maintain a priority queue with all peers and their loads, and for each node u we successively pop the lightest peer p and we re-insert it with its new load-value `Load`(p)+`Load`(u), due to the node assignment of u to p. Whether we select i at random, or due to its heavy load, affects only the number of iterations that the process will need in order to converge.

Algorithm 2. Inflationary balancing algorithm

```
1: repeat
2:     state = State()
3:     hosts = minHeap()
4:     nodes = maxHeap()
5:     h = heaviest()
6:     for i in h.routingTable do
7:         for j in i.nodes do
8:             nodes.push((l_j, j))
9:         end for
10:        hosts.push((0, i))
11:        cache(state, i)
12:    end for
13:    for j in h.nodes do
14:        nodes.push((l_j, j))
15:        hosts.push((0, h))
16:        cache(state, h)
17:    end for
18:    while not nodes.empty() do
19:        (L_i, i) = hosts.pop()
20:        (l_j, j) = nodes.pop()
21:        hosts.push((L_i + l_j, i))
22:        assign(j, i)
23:    end while
24: until allocationChange(state)
```

The *replication* technique aims at alleviating bottlenecks, by imposing additional redundancy, and distributing load of hotspots among more than one hosts, enhancing high availability and fault tolerance. Obviously, this method aims exclusively at task-skew. Our approach, instead of replicating single popular data tuples, focuses on how we can replicate nodes, manage them efficiently, and preserve consistency among all replicas of an overlay node. When replicating hotspots, only a portion of the original node's traffic reaches each copy, and as a result heavy nodes are alleviated. According to our paradigm, we replicate nodes responsible for popular areas to as many hosts is needed, so that there is no task-skew among hosts. Hence, replication factor varies from node to node, with respect to their load. Our *local allocation* is based exclusively on a peer's limited knowledge about the network. At each iteration we select a random peer and we redistribute all linked peers to the associated nodes. This is an extremely flexible and effective solution as we assign hosts to nodes in such a way that overlay load imbalances render obsolete (one-to-many).

Algorithm 3. Deflationary balancing algorithm

```
 1: repeat
 2:     state = State()
 3:     hosts = Vector()
 4:     nodes = minHeap()
 5:     h = heaviest()
 6:     for j in h.routingTable do
 7:         hosts.expand(replicas(j))
 8:         nodes.push((0, j))
 9:         cache(state, j)
10:     end for
11:     hosts.expand(replicas(h))
12:     nodes.push((0, h))
13:     cache(state, h)
14:
15:     while not hosts.empty() do
16:         (L_j, j) = nodes.pop()
17:         assign(j, hosts.pop())
18:         nodes.push((l_j / len(j.hosts), j))
19:     end while
20: until allocationChange(state)
```

4 Experimental Evaluation

In order to assess our methods and evaluate their performance, we performed extensive experiments with workloads of varying dimensionality and skewness.

4.1 Setting

Our experimental evaluation consists of two major parts. A static part, whereby different allocations for rigorous balancing techniques are compared, while the latter part

consists of dynamic simulations. In the former part, we are especially interested on how our paradigm performs compared to other types of allocation. Most of the prior works in balancing use a straightforward *naive* approach for random and arbitrary assignment of balls to bins. Consequently, no special consideration is made on how peers allocate nodes. Nevertheless, we are also interested to compare our methods with an *ideal* allocation for each method. More specifically, we make use of a heuristic, greedy allocation based on global knowledge, according to which each time the heaviest ball is assigned to the globally lightest bin; whereas our methods rely exclusively on peer knowledge. Albeit on-line, the aforementioned allocation type is not realistic as each peer has only partial knowledge of the network, substantially small but functional.

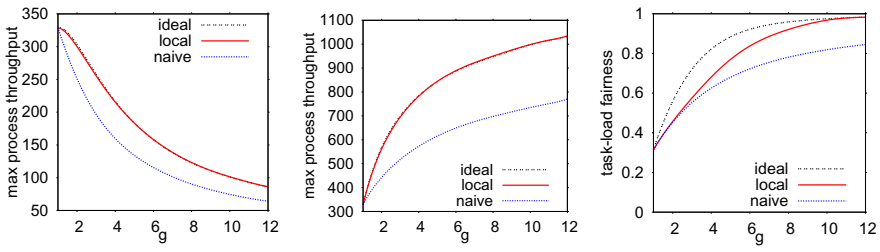
In all experiments, we make use of the PGrid-Z [3] overlay, which incorporates a Z-curve into P-Grid to support multidimensional range search, and we evaluate our methods by various metrics. Latency is the maximum distance in terms of hops from the initial peer to any peer reached during search. Albeit, maximum process throughput $\Lambda_{\max}$ [3] may be criticized as being too pessimistic, as it only depends on the single most loaded peer, it can be argued that just one overloaded peer will indeed cause trouble for the rest of the network in practice, as being a bottleneck will affect network delay and drop requests in a struggling effort to cope with overwhelming traffic. We are also interested in storage and task load fairness. In particular, we use Jain's fairness index [5], to measure data- and task-skew. Furthermore, we consider successful a method that is capable of exploiting redundancy to enhance performance. Therefore, we measure redundancy from the most replicated node. In addition, the maximum number of comprised nodes in a single peer is also a significant metric, as a peer that contains many nodes has to deal with high data load and maintenance cost. We also present the communication cost in terms of exchanged data tuples.

We use of datasets describing roads and rivers of Greece. The final dataset consist of 100K tuples. Each queryset consists of 10K range queries, as this type of search is among the most resource consuming. In particular, we define three cluster-centers corresponding to the largest cities of Greece: Athens, Thessaloniki and Patras. All clusters follow normal distributions and querysets consist of queries generated from a cluster chosen with probability proportional to the population of each town. Last, we also created synthetic datasets of variable dimensionality to study the impact of dimensionality.

4.2 Results

On the whole there is a significant benefit from using our methods instead of using the straightforward naive approach. Figure 1 illustrates the efficiency of our methods, as ideal allocations, based on global overlay knowledge, perform only slightly better compared to our schemes that rely exclusively on a peer's partial knowledge about the overlay. The *multiple realities* technique is unsuitable for complex operations where result is returned in fragments over time. Specifically, as shown in Figure 1a, this is the only method where max process throughput $\Lambda_{\max}$ diminishes with g for all allocation types. Evidently, enacting a request in all realities in order to improve latency impairs $\Lambda_{\max}$ dramatically. Most importantly, it fails to improve latency enough. Contrary to what expected, latency was only slightly affected by the number of multiple realities because this technique was designed for lookup queries. In Figure 1b, our *parallel universes*

technique outperforms the former technique in terms of $\Lambda_{\max}$ and shows an increasing behavior with respect to the number of realities. Clearly, this method has a significant impact on max process throughput. Moreover, we had to compare our paradigm with different allocation methods. Regarding naive allocation, we assume for each reality, nodes are mapped to arbitrary peers. For the ideal allocation we consider again a centralized method that is run for each reality. Specifically, we assign the heaviest node to the lightest peer with respect to a peer's summing load in all realities preceding the one being examined. Then, each peer comprises exactly g nodes, each participating in a different reality. In addition, task-load fairness index ameliorates with g (Fig. 1c).



(a) multiple realities **(b)** parallel universes **(c)** parallel universes

(d) replication **(e)** virtual nodes **(f)** virtual nodes

Fig. 1. Max process throughput and task-load fairness

Regarding the *virtual nodes* technique, $\Lambda_{\max}$ shows a concave behavior in Figure 1e, due to a combination of two phenomena. For low g values, where fragmentation is insignificant, fairness beneficial effects are dominant, and thus, $\Lambda_{\max}$ ameliorates. However, it diminishes as routes become longer for higher g values. However, congregating numerous nodes increases maintenance costs. Apart from keeping multiple routing tables, peers are burdened with maintenance tasks, such as detecting failures. In addition, latency and precision are affected due to the method's direct interaction with the overlay size. Latency increases with g as larger overlays result in longer message routes. On the other hand, skewness helps precision because the query clusters were imported in dense areas of the key-space. Hence, when peers join the network by selecting a key with equal probability (data balanced), they populate densely those areas as well. In addition, precision increases with g, as the overlay nodes become responsible for smaller areas, while our queries have fixed size, and thus, less irrelevant nodes become reached. In effect,

this method infuses the overlay with $\Lambda_{\max}$, precision and latency from larger overlays with respect to g. Concerning the naive assignment of nodes to peers, we adopt the randomized strategy for each node to be assigned to any peer with equal probability. For, the ideal centralized allocation scheme the heaviest node is assigned greedily to the globally lightest peer at the time. Clearly, as Figure 1e depicts, our approach outperforms naive assignements as it provides 3.5 times better $\Lambda_{\max}$. More importantly, ideal allocations are less that 5% more efficient than our scheme for all configurations and g values.

For *replication*, the greater the redundancy, the less traffic bottlenecks intake. Thereby, imbalances are blunted as overloaded peers are alleviated. Naturally, maximum redundancy increases linearly with g until fairness is achieved, and withal, it reaches an upper bound beyond which there is no further benefit in load fairness index. Hence, we limit g to take values smaller than this value. In addition, the invariant latency can be explained as routing takes place along the overlay and not the actual network. Concerning competitor schemes, for the naive allocation, each host replicates a randomly selected node with equal probability. The ideal allocation is a centralized algorithm that probes in sequence all nodes and each time copies the heaviest node to an available host. Figure 1d shows that there are immense benefits of using our method instead of the corresponding naive allocation. It also reveals that this is the most efficient technique in terms of $\Lambda_{\max}$, as it selectively replicates hotspots.

5 Conclusions

To recapitulate, the virtual nodes technique improves performance for certain configurations due to its combined effects, without imposing additional redundancy. Nonetheless, this method accumulates data load from all comprised nodes. The multiple realities technique is considered appropriate exclusively for lookup queries as complex queries impair performance dramatically; whereas our parallel universes paradigm enhances performance significantly. Replication is a flexible method that does not impose fixed additional redundancy, and allows the network to adapt to any load distribution.

References

1. Avidor, A., Azar, Y., Sgall, J.: Ancient and new algorithms for load balancing in the p norm. Algorithmica 29(3), 422–441 (2001)
2. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: supporting scalable multi-attribute range queries. In: SIGCOMM, pp. 353–366 (2004)
3. Blanas, S., Samoladas, V.: Contention-based performance evaluation of multidimensional range search in p2p networks. In: InfoScale 2007, pp. 1–8 (2007)
4. Ganesan, P., Bawa, M., Garcia-molina, H.: Online balancing of range-partitioned data with applications to peer-to-peer systems. In: VLDB, pp. 444–455 (2004)
5. Jain, R., Chiu, D., Hawe, W.: A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. DEC TR-301 (1984)
6. Karger, D.R.: Simple efficient load balancing algorithms for peer-to-peer systems. In: ACM SPAA, pp. 36–43 (2004)
7. Rao, A., Lakshminarayanan, K., Surana, S., Karp, R.M., Stoica, I.: Load balancing in structured p2p systems. In: IPTPS, pp. 68–79 (2003)
8. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: SIGCOMM 2001, pp. 161–172 (2001)