Computing (2012) 94:731–762
DOI 10.1007/s00607-012-0199-7

A novel robust on-line protocol for load-balancing in structured peer-to-peer systems

George Tsatsanifos · Vasilis Samoladas

Received: 16 November 2011 / Accepted: 9 June 2012 / Published online: 13 July 2012

Abstract In this paper, we revisit the problem of load-balancing structured peer-to-peer systems with on-line protocols. Load-balancing is of major significance for large-scale decentralized networks in terms of enhanced scalability and performance. The main incentives behind balancing schemes are under-utilization of bandwidth and computer resources. Therefore, our methods focus mainly on task-skew. Specifically, we address the problem with on-line protocols on the basis of migration and enhanced availability. In particular, the cornerstones of our methods are the notions of *virtual nodes*, *replication* and *multiple realities*, combined altogether with allocation techniques based on *balls-in-bins games*. The rationale of our dynamic protocol to depend exclusively on peer load distribution preserves intact the structural properties and search efficiency of the overlay used as an indexing infrastructure, while preserving the semantic information of the data (e.g., range partitioned network). We also propose an effective load-aware mechanism to facilitate robust operations that counteract against contingent churn failures. Finally, our work is complemented with extensive experiments using both real and synthetic data sets.

Keywords Load-balancing · Structured peer-to-peer systems · Distributed hash-tables · Multiple realities · Virtual nodes · Replication · Range queries

Mathematics Subject Classification 68M10 · 68M12 · 68M14

G. Tsatsanifos (✉)
Knowledge and Database Systems Laboratory,
School of Electrical and Computer Engineering,
National Technical University of Athens, Athens, Greece
e-mail: gtsat@dblab.ece.ntua.gr

V. Samoladas
Software Technology and Network Applications Laboratory,
Electronic and Computer Engineering Department,
Technical University of Crete, Chania, Greece
e-mail: vsam@softnet.ece.tuc.gr

1 Introduction

Load-balancing has never been an easy problem for computer science. It is a challenging open problem with theoretical aspects directly related to well-known NP-complete problems (partition, makespan) and combinatorial optimization issues, as well. These problems are approached through approximation due to their complex nature. The main incentives behind balancing schemes are under-utilization of bandwidth and computer resources. Besides, load-balancing is a real problem with practical extensions on many system parameters, pertaining to optimal resource utilization, throughput, and response time. Additionally, load imbalances may cause severe impediments, such as bottlenecks and congestion, which are critical to an effective network operation. Moreover, dynamically changing popularities of data items and skewed query patterns in peer-to-peer systems may cause some of the peers to become bottlenecks, thereby resulting in severe load imbalance and consequently increased response times.

In this paper, we revisit the problem of load-balancing structured peer-to-peer systems [33,40–42,47]. In principle, structured peer-to-peer systems are large-scale decentralized networks that employ a global consistent protocol that addresses the following aspects: (i) a partitioning protocol of the key-space, such as a multi-dimensional interval, so that each peer is responsible for a specified key-space partition. Then, a peer responsible for a particular key-space partition should store all (key,value) pairs that correspond to its zone, (ii) a graph embedding among these partitions (peers) which guarantees the existence of at least one path from any peer to all others, (iii) a routing protocol that enables message forwarding in such a way that a query spanning a specific part of the interval will reach all peers whose zones overlap with the query in a number of hops smaller than the diameter of the graph, and (iv) a fault-tolerance mechanism that ensures that all the above hold as peers join, leave and fail arbitrarily. Notably, the sheer size of peer-to-peer systems and the inherent dynamism of the environment pose significant challenges to load-balancing. Moreover, the main difficulty in supporting complex queries in a peer-to-peer system, such as range search, while providing load balancing is that the order of elements within the data structure must be preserved. Our intention is to develop a realistic balancing paradigm, which can be used on top of any DHT overlay, to mitigate load imbalance effects, and thereby, enhance performance and scalability. Our methods are focused mainly on task-skew. We address this multifaceted problem with rigorous protocols based on replication and migration, reducing the problem to a balls-and-bins game by facing hosts as bins and virtual nodes as balls.

The contributions of this paper are multifold. In the first part of this work, we study rudimentary balancing techniques, namely multiple realities, virtual nodes and replication 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. In the latter part of our work, we present a balancing protocol that compromises the tradeoffs between the options of migration and replication, dynamically adjusts the number of replicas and virtual nodes, and adapts their allocation to peers in response to network changes. Our protocol combines desired characteristics from each technique, though, from a new point of view that takes seriously balls-in-bins games into consideration. In effect, this

means that some parts of the network might use the virtual nodes technique, while some nodes of the overlay might be replicated by many hosts in the network at the same time. More importantly, our scheme can be applied on top of any overlay without damaging its properties (e.g., processing complex queries), regardless indexing scheme, dimensionality degree. This is a salient characteristic of this work, since most of the prior relevant works in the literature have concentrated on topology re-organization efforts that rely on transferring data between adjacent nodes, and thus, making their balancing objectives infeasible for distributed multi-dimensional data structures. To the best of our knowledge, our protocol is the first to unify these techniques into a single protocol for structured peer-to-peer systems. Moreover, by taking advantage the structural properties of our scheme, we are in position of augmenting our work with an effective load-aware mechanism to ensure resiliency against contingent churn failures. Besides, fault-tolerance is an indispensable constituent of a complete solution for large-scale decentralized networks, where peers join, leave and fail arbitrarily. Finally, our experimental evaluation demonstrates that our proposed techniques are indeed effective in improving the system performance significantly.

The remainder of this paper is organized as follows. Section 2 augments rudimentary balancing algorithms. In Sect. 3 we describe our dynamic balancing protocol. Section 4 discusses an augmented fault-tolerance mechanism. Section 5 presents thorough experiments and evaluates our work. Section 6 reviews relevant literature. Last, Sect. 7 summarizes and concludes our work.

2 Balanced allocations from a new perspective

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. In particular, we apply the balls-in-bins model to the following techniques:

- Maintaining multiple, independent coordinate spaces with each node being assigned a different zone in each coordinate space.
- Associating keys with virtual nodes and mapping multiple virtual nodes to hosts.
- Replicating popular nodes into many hosts, sharing this way hotspots' load.

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. Additionally, a peer may enact a request via any of its congregated nodes.

In retrospect, 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 contribution involves the design of iterative allocation schemes based exclusively on local knowledge that exploit balls-in-bins games with minimum makespan scheduling approximate solutions [10,25,37]. 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.

2.1 Virtual nodes

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. In essence, n hosts allocate the $m = g \times n$ nodes of the overlay, where $g > 1$, and thus, this method results in assigning many nodes to a single peer; unlike alternative balancing methods that exploit enhanced availability. Intuitively, it offers a tantalizing opportunity for balancing peers' data-load by providing a more uniform coverage of the identifier space. This does not affect the worst-case path length. However, the maintenance cost for a peer congregating many nodes (e.g., updating routing tables) increases readily. Furthermore, 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. More importantly, the virtual nodes technique is devoid of data-redundancy, as a peer may contain many nodes but each node can be hosted in one peer exclusively (many-to-one scheme).

Specifically, the maximum load of the busiest peer equals the sum of the loads of all its comprised nodes, and thus, can never be less than the load of the busiest node. This, naturally, constitutes a barrier for this method regarding task-load fairness which can be faced with the virtual nodes method. However, if we opt for iteratively splitting the heaviest node until an assignment of nodes to peers can be found until some criterion is met, we cannot be sure about the outcome of such operation as beyond some point, too much fragmentation has negative consequences. We also note that the increased maintenance cost, arising from comprising a large number of virtual nodes, involves an additional communication cost that has the opposite result, and could have been avoided otherwise. As a result, if there are too many virtual nodes compared to the number of peers, then max task-load might increase instead of improve, as we will show next in our evaluation part. Thereby, we can intuitively understand that there is some "golden" ratio for the number of virtual nodes in terms of the available peers.

2.1.1 Analysis of the centralized scheme

In this section we are in search for a compact and instructive bound on the maximum peer load for the centralized greedy allocation of nodes to peers. Let m the number of peers and n the number of virtual nodes with $n = g \times m$. In addition, let Q be the total

number of queries issued by all nodes of the overlay. We assume in our example that all queries are either lookup or range queries and therefore each requires $O(\log_2 n)$ hops and messages for most of the overlay structures in the literature, e.g. Chord [42], P-Grid [4], VBI-Tree [27], BATON [26], MIDAS [43], Pastry [41], Tapestry [47], and many others. Hence, a total of $Q \log_2 n$ messages at most is forwarded throughout the overlay until all queries are resolved. Now, let μ_j the load of virtual node j. In our case load corresponds to the number of requests that node j receives (task-load). Hence, we have for the average load that,

$$\mu_{\text{mean}} = \frac{1}{n} \sum_{j=1}^{n} \mu_j \leq \frac{1}{gm} Q \log_2 gm \tag{1}$$

Let M_i the total load of peer i, which equals to the sum of the loads of all comprised nodes in peer i, $M_i = \sum_{j \in i} \mu_j$. Let p the heaviest peer produced by the algorithm, and let v be the index of the last virtual node assigned to this peer. Next, let $M_{p \setminus v}$ be the load of peer p before virtual node v gets assigned to peer p. Since the algorithm assigns a job to the least loaded peer at the time, it follows that all peers have loads greater than $M_{p \setminus v}$. This implies that,

$$M_{p \setminus v} = \frac{1}{m} \sum_{\substack{j=1 \\ j \neq v}}^{n} \mu_j$$

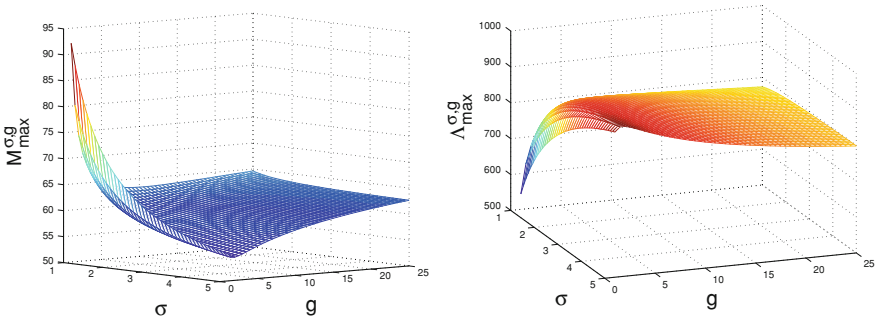
$$M_{p \setminus v} \leq \frac{1}{m} (Q \log_2 gm - \mu_v)$$

Moreover, the heaviest peer's load is at most $M_p = M_{p \setminus v} + \mu_v$, where μ_v corresponds to the load of the lightest virtual node being the last node being assigned to a peer from a sorted list, $\mu_v \leq \mu_j, \forall j$. Then, we are in position of coming up with an upper bound for M_p after the greedy assignment of virtual nodes to peers. Henceforth, we will refer to the upper bound for the load of the heaviest peer as $M^g_{\max}$. Note that we regard $M^g_{\max}$ as a continuous function of g, with $g \geq 1$, and therefore we are in search for the g-value(s) that minimize $M^g_{\max}$. More specifically, we provide a tight example according to which the worst case scenario for the heaviest peer is shown, as for any other allocation of virtual nodes M_p intakes a lighter load. Therefore, we want to find the g-values such that, $g_{\text{opt}} = \text{argmax}_g M^g_{\max}$. Hence, when replacing and summing Eqs. 1 and 2 we have that,

$$M^g_{\max} = \frac{1}{m} Q \log_2 gm + \frac{m-1}{m} \mu_{\min} \tag{2}$$

More importantly, the $\mu_{\min}$ factor introduces skewness which affects $M^g_{\max}$ and the g-values where it is minimized. In particular, we let σ signify how many times the average node's load is greater than the lightest node.

$$\mu_{\min} = \frac{\mu_{\text{mean}}}{\sigma} \leq \frac{1}{\sigma gm} Q \log_2 gm, \quad \text{where } \sigma \geq 1 \tag{3}$$



(a) Maximum task-load, $M_{\max}$. **(b)** Maximum process throughput, $\Lambda_{\max}$.

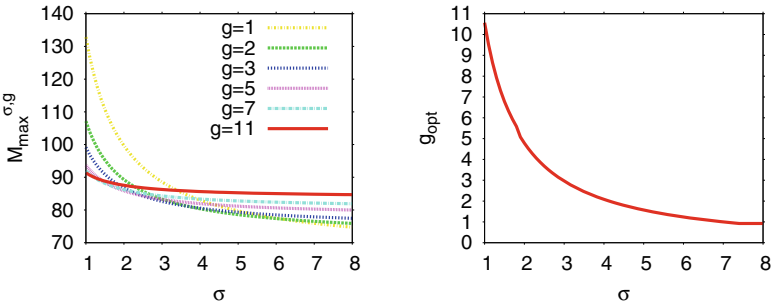
Fig. 1 Performance upper bounds in terms of the virtual nodes to peers ratio g for $10K$ peers, $50K$ queries, and varying task-load skewness σ

Therefore, Eq. 2 now becomes

$$M_{\max}^{\sigma,g} = \frac{Q}{m} \log_2 gm \left(1 + \frac{m-1}{\sigma gm}\right) \tag{4}$$

From Eq. 4 it follows that $M_{\max}^{\sigma,g}$ is a function of both σ and g. Now we are in position of finding the optimum configuration for g. In effect, these are the values in Fig. 1a that minimize $M_{\max}^{\sigma,g}$ for fixed skewness σ and a fixed number of peers m.

Quite notably, hitherto approaches evaluate imbalances by the maximum number of virtual nodes congregated into a single peer, whereas our analysis takes also into consideration the load of the comprised nodes. Furthermore, by establishing an analysis regarding the maximum load of the heaviest peer in a peer-to-peer network, we can predict the maximum possible process throughput in the network, as proposed in [14]. This is an importance performance metric which quantifies the resilience of a peer-to-peer network to contention by concurrent searches. More formally, assume that each peer can process at most one message per unit of time. If the queries from the workload arrive (stochastically) at a rate of Λ queries per unit of time, each query with equal probability, then messages to peer i will arrive at a rate of $\frac{\Lambda M_i}{Q}$ messages per unit of time. Assuming that peer i is not overloaded (messages do not arrive faster than it can process them) we have $\Lambda < \frac{Q}{M_i}$. Then, maximum process throughput $\Lambda_{\max}$ is defined as the maximum value of Λ so that no peer is unduly loaded $\Lambda_{\max} = \frac{Q}{\max_i M_i}$. Consequently, when Λ exceeds $\Lambda_{\max}$ the overlay is burdened with requests it can no longer process and thus the network becomes unstable, performance impairs, and some portion of the queries that cannot be processed are possibly lost. Figure 1b depicts the maximum process throughput for the same configuration. Most importantly, the g-values where $M_{\max}^{\sigma,g}$ is minimized, naturally, coincide with the values that maximize $\Lambda_{\max}$. Moreover, $M_{\max}$ also depends on query-load Q, whereas Λ, which expresses the maximum possible number of queries that can be processed per time unit, is normalized and can be compared with the corresponding maximum process throughput from other overlays for different workloads even.



(a) Max task-load for different ratios of nodes to peers.

(b) Optimum values for the ratio of virtual nodes to peers.

Fig. 2 $M_{\max}$ and the g-values where it is minimized for $10K$ peers, $50K$ queries, and varying task-load skewness σ

Then, the optimal g-values can be found by solving the equation $\frac{d}{dg}M_{\max}^{\sigma,g} = 0$ in terms of g. Hence, we need first to differentiate Eq. 4 which leads to,

$$\frac{d}{dg}M_{\max}^{\sigma,g} = \frac{Q}{m\ln 2}\frac{\sigma mg + (m-1)(1-\ln mg)}{\sigma mg^2} \tag{5}$$

Due to its form, we solve the equation $\frac{d}{dg}M_{\max}^{\sigma,g} = 0$ that returns the optimal configuration using a well known numerical analysis method, namely Newton's method, according to which we start from an arbitrary solution which we improve sequentially. Hence, starting from an initial value $g^{(0)}$, we have for the $k+1$th iteration,

$$g^{(k+1)} = g^{(k)} - \frac{\frac{d}{dg}M_{\max}^{\sigma,g^{(k)}}}{\frac{d^2}{dg^2}M_{\max}^{\sigma,g^{(k)}}} \tag{6}$$

Then, by differentiating Eq. 5 we take that,

$$\frac{d^2}{dg^2}M_{\max}^{\sigma,g} = -\frac{Q}{m\ln 2}\frac{\sigma mg + 3(m-1) - 2(m-1)\ln mg}{\sigma mg^3} \tag{7}$$

Hence, it follows that,

$$g^{(k+1)} = g^{(k)} + g^{(k)}\frac{\sigma mg^{(k)} + (m-1)(1-\ln mg^{(k)})}{\sigma mg^{(k)} + 3(m-1) - 2(m-1)\ln mg^{(k)}} \tag{8}$$

We choose to terminate this iterative procedure when $|g^{(k+1)} - g^{(k)}| < \epsilon$, where $\epsilon \to 0$, which converges in just a few iterations. In Fig. 2a we present $M_{\max}$ in terms of varying skewness σ for different g-values. To elaborate, Fig. 2a suggests for each value of the skewness factor the configuration for g which is below all others as the fittest. Succinctly, Fig. 2b provides an insight of how to optimize performance and minimize

the load of the heaviest peer in a network and also maximize process throughput, as g_{opt} designates what the optimum configuration is for varying skewness. As a result, g_{opt} constitutes the maximal meaningful values for g, as configurations beyond g_{opt}, apart from being sub-optimal in terms of performance, but also include additional costs for purposes like maintenance, updating routing tables, etc. Nonetheless, for higher g-values still high-levels of fairness are achieved. Besides, load-balancing is a multi-faceted problem, and above all, performance is its most conspicuous aspect. We reckon that this is a remarkable new result which sheds new light on load-balancing for peer-to-peer systems. Specifically, it had been obscured by hitherto approaches that focused on the maximum number of virtual nodes into a single peer by assuming that virtual nodes have equal loads. It now becomes obvious that an increased number of virtual nodes cannot necessarily guarantee stellar performance. A final note regards extremely skewed load distributions in the overlay, where increasing the number of virtual nodes with respect to the available peers does not seem to improve $M_{\max}$. In fact, with increased skewness σ, we observe that g_{opt} diminishes.

2.1.2 Analysis of the LBB algorithm

Our local allocation scheme which exploits the virtual nodes technique, is limited to peer knowledge about the overlay. Algorithm 1 constitutes the balancing process which has been triggered by peer x, either to insert a new peer y, or because x has traced its being overloaded compared to its neighbors, and therefore, invokes a balancing procedure among all peers in i's routing table (for the latter case though, we omit line 1 from Algorithm 1). Peer x is aware of the loads of its neighboring peers and their

Algorithm 1 *joinLBB1:* the *LBB* algorithm that exploits *virtual nodes*

```
1: T ← getAverageNeighborhoodLoad()
2: nph ← createMinPriorityPeerHeapFromUnderLoadedPeers(T)
3: nph.add(n)   // take into account the new peer n as well
4: while not nph.isEmpty() do
5:   p ← nph.pop()
6:   G_p ← |T − p.getLoad()|
7:   xph ← createMaxPriorityPeerHeapFromOverloadedPeers(T)
8:   while not xph.isEmpty() do
9:     q ← xph.pop()
10:    G_q ← |q.getLoad() − T|
11:    while q.getNodesNr() > 1 do
12:      u ← argmin_{v∈q} |v.getLoad() − G_p|
13:      if |G_p − v.getLoad()| < G_p and |G_q − v.getLoad()| < G_q then
14:        p.add(v)
15:        q.remove(v)
16:        G_p ← |G_p − v.getLoad()|
17:        G_q ← |G_q − v.getLoad()|
18:      else
19:        break
20:      end if
21:    end while
22:  end while
23: end while
```

comprised virtual nodes and therefore is able to compute the average neighborhood load T (line 1). Next, x creates a max priority heap with all neighboring peers whose loads exceed T (line 1), and a min priority heap with all neighboring peers with loads less than T (line 1). Additionally, if peer x was contacted by a new peer y that wants to participate in the network, then y will also be included in the underloaded peers' heap as well (line 1) with load equal to zero, otherwise this line is omitted. Now, for each of the underloaded peers (line 1), starting from the lightest, we find the node that fits best its load gap from T (line 1) from the heaviest peer with more than one virtual nodes (lines 1, 1), which is then moved from the heavy to the light peer (lines 1, 1). Load-gap values are updated (lines 1, 1) and we repeat until the load-gaps of the light and the heavy peers do not improve anymore, and then we proceed with the next underloaded peer from the heap. On the other hand, Algorithm 2 presents the balancing process triggered by departing peer p, which assigns sequentially its nodes, starting from the lightest, to its lightest linked peer.

We will now establish a similar analysis for a tight example regarding the *LBB* Algorithm in order to establish a necessary upper bound $M_{\max}$ for task-load for the most loaded peer. Again, we have m peers and n virtual nodes. Initially, we assume that peers contain for connectivity purposes just one virtual node with load equal to ϵ, where ϵ is a arbitrarily small quantity; except for one peer that is responsible for $g \times m$ virtual nodes, and as a result, it is duly loaded. We note that each peer has information for at least $\log m$ other peers. Therefore, the overloaded peer has to share its load with all associated peers. These are the peers that its virtual nodes' links are hosted. In turn, these peers will attempt to balance their load with their own associated peers, and so on. Hence, an hierarchy is formed, according to which the overloaded peer is on top. In particular, peers at level j are j hops away from the overloaded peer. More formally, we assume a function ϕ_j which returns the minimum number of *newly* inserted peers at depth j of the hierarchy. Then, $\phi_0 = 1$ and ϕ_1 corresponds to the number of distinct peer entries in the routing table of the overloaded peer. In particular, for trie-structured overlays, like P-Grid [4] and MIDAS [43], we have for our example that $\phi_j = 2^j(\log m - j) - 2^j + j$.

Algorithm 2 *departLBB1:* the *LBB* algorithm that exploits *virtual nodes*

```
1: xnh ← createMaxPriorityNodeHeapFromLocalNodes()
2: nph ← createMinPriorityPeerHeapFromLinkedPeers()
3: while not xnh.isEmpty() do
4:   u ← xnh.pop()
5:   if not nph.isEmpty() then
6:     q ← nph.pop()
7:     q.add(v)
8:     p.remove(v)
9:     nph.push(q)
10:  end if
11: end while
```

We will now establish a case study for $M_{\max}$ and how it diminishes according to the virtual nodes paradigm for a single-point bottleneck example. This procedure can be described by a series of phases, where each phase corresponds to an effort for each

peer of the hierarchy, starting from the top, to balance its load with its direct neighbors that correspond to the lower level. More importantly, we will investigate how load disseminates from a single hotspot to the rest of the network. In particular, since $M_{\max}$ is expressed as a sum of two parts, namely M_p and μ_v, as in Eq. 2, we will first study the course of $M_p^{i,j}$ for each level j of the hierarchy in phase i under the assumption of fractional peer-load. Then, on top of $M_p^{i,j}$ we will add the appropriate node load quantity μ_v in order to complete our worst case analysis.

Initially, at phase 0 the overloaded peer's load equals to the total load of the overlay, say $M_p^{0,0}$, while all other loads are very small, with $\epsilon \to 0$. At phase 1, each peer-load at level 1 now equals to $M_p^{1,1} \leftarrow M_p^{0,0} \frac{\phi_0}{\phi_0+\phi_1}$, since part of the overloaded peer's load is shed to its ϕ_1 direct neighbors, and thus, now $M_p^{1,0} \leftarrow M_p^{1,1}$. Next, each of the ϕ_1 previously encountered peers will contribute approximately $\frac{\phi_2}{\phi_1}$ new peers each, and thus, each peer will invoke a balancing procedure among $\frac{\phi_2}{\phi_1} + 1$ peers and their congregated nodes. As a result, each peer at level 2 will be responsible for nodes that correspond to load equal to $M_p^{1,2} \leftarrow M_p^{1,1} \frac{\phi_1}{\phi_1+\phi_2} = M_p^{0,0} \frac{\phi_0}{\phi_0+\phi_1} \frac{\phi_1}{\phi_1+\phi_2}$, and then we have that $M_p^{1,1} \leftarrow M_p^{1,2}$. Likewise, we find how load disseminates from a single hotspot to the remaining levels according to the formula $M_p^{1,j} \leftarrow \prod_{k \leftarrow 0}^{j-1} M_p^{0,0} \frac{\phi_k}{\phi_k+\phi_{k+1}}$. Hence, this first phase ends when some part of the load of the peer at the top of the pyramid reaches the peers at the bottom. Next, at phase 2 we will repeat the same procedure. From now on however, at each balancing procedure the loads that were assigned to the previous phase will have to be included, as well. Hence, the peer's load at level 1 equals to $M_p^{2,1} \leftarrow \frac{\phi_0 M_p^{1,0} + \phi_1 M_p^{1,1}}{\phi_0+\phi_1}$, and then, we set $M_p^{2,0} \leftarrow M_p^{2,1}$. Next, each peer's load at level 2 will now be equal to $M_p^{2,2} \leftarrow \frac{\phi_1 M_p^{2,1} + \phi_2 M_p^{1,2}}{\phi_1+\phi_2}$. Generally, following the dynamic programming principles, we have for layer j at phase i that $M_p^{i,j} = \frac{\phi_{j-1} M_p^{i,j-1} + \phi_j M_p^{i-1,j}}{\phi_{j-1}+\phi_j}$. In other words, load gradually flows at each phase from the top layers to the lower levels of the hierarchy that are more distant from the overloaded peers, until fairness is achieved among the different levels. Then, it is not hard to show that after a finite number of phases i, $M_p^{i,j}$ converges for all depths j to the same M_p quantity. Nonetheless, on top of M_p one cannot add $\mu_{\min}$, as we did in Eq. 2, since one cannot be certain about the position/level of the heaviest peer due to the numerous independent balancing operations all over the hierarchy. Hence, the load of the node assigned last cannot be predicted. However, at all cases we know that it is $\leq \mu_{\text{mean}}$, and thus, it follows that $M_{\max}^g = M_p + \mu_{\text{mean}}$, which is essentially greater or equal than the corresponding bound for the centralized approach. As expected the centralized assignment of virtual nodes to peers dominates the distributed algorithm, and therefore, it will be used as a measure of optimality which can indicate whether a greedy distributed allocation can ever be just as good.

2.2 Redundant overlays

The principle of this 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. For the *parallel universes* technique we want all peers to be burdened with equal loads due to their participation in all redundant overlays, since balancing is our main concern before latency. To attain our objectives, we force peers to enact their queries only in one of the realities, selected randomly and independently. In particular, we consider for our scheme the case where n peers participate in the network in all g parallel overlays, where each consists of $g \times m$ nodes. Therefore, in order to balance, we reassign the nodes from all realities of a selected peer's nodes' overlay vicinities (links) in such a way that all peers acquire nodes with approximately the same summing loads. Even though the query task-load is immensely reduced, data insertions and deletions still have to be applied to all realities, otherwise the system is inconsistent, in that identical requests in different realities would yield different results. However, this scheme comes at a cost of a fixed replication factor. In effect, along with overweight areas of space, the underweight ones are replicated as well.

In retrospect, the balancing scheme that was endorsed in the CAN paper [40], namely the multiple realities paradigm, also leverages redundant overlays from a completely different point of view, though. This method aims at reducing latency instead by invoking queries simultaneously in all realities and get the fastest answer. When a peer simultaneously "broadcasts" its request to all realities, the fastest answer is returned to the user, and thus latency ameliorates. Hence, when a lookup query is enacted simultaneously in all g realities, it requires g times more messages to be resolved. No other consideration has been made, other than creating and maintaining redundant overlays.

Algorithm 3 *joinLBB2:* the *LBB* algorithm that exploits *multiple realities*

```
1: for j ← 1 to realitiesNr do
2:   T_{1→j} ← getAverageNeighborhoodLoad(1 → j)
3:   nph ← createMinPriorityPeerHeapFromUnderLoadedPeers(T_{1→j})
4:   nph.add(n)   // take into account the new peer n as well
5:   while not nph.isEmpty() do
6:     p ← nph.pop()
7:     G_p ← |T_{1→j} − p.getLoad(1 → j)|
8:     xph ← createMaxPriorityPeerHeapFromOverloadedPeers(T_{1→j})
9:     while not xph.isEmpty() do
10:      q ← xph.pop()
11:      G_q ← |q.getLoad(1 → j) − T_{1→j}|
12:      while q.getNodesNr(j) > 1 do
13:        u ← argmin_{v∈q} |v.getLoad() − G_p|
14:        if |G_p − v.getLoad()| < G_p and |G_q − v.getLoad()| < G_q then
15:          p.add(v, j)
16:          q.remove(v, j)
17:          G_p ← |G_p − v.getLoad()|
18:          G_q ← |G_q − v.getLoad()|
19:        else
20:          break
21:        end if
22:      end while
23:    end while
24:  end while
25: end for
```

Algorithm 4 *departLBB2:* the *LBB* algorithm that exploits *multiple realities*

```
1: for j ← 1 to realitiesNr do
2:   xnh ← createMaxPriorityNodeHeapFromLocalNodes(j)
3:   nph ← createMinPriorityPeerHeapFromLinkedPeers(j)
4:   while not xnh.isEmpty() do
5:     u ← xnh.pop()
6:     if not nph.isEmpty() then
7:       q ← nph.pop()
8:       q.add(v, j)
9:       p.remove(v, j)
10:       nph.push(q)
11:     end if
12:   end while
13: end for
```

More specifically, our local allocation method, *LBB*, enhances our parallel universes scheme and 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. In Algorithm 3, variable $T_{1\rightarrow j}$ stands the average peer load in the neighborhood in the first j universes. The data-redundancy introduced in both schemes is fixed and equal to the number of realities g.

2.3 Replication schemes

The *replication* technique aims at alleviating bottlenecks, by imposing additional data-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. This approach resembles very much the virtual nodes technique in reverse, though. Instead of making a peer responsible for many unrelated parts of the key space, we make a popular area available through many peers. 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).

A natural question is raised regarding the redundancy imposed and the involved costs. We measure this by the cardinality of the most replicated overlay node. More formally, let $\mu_{\max}$ the load of the most loaded node of the overlay, with $\mu_{\max} = \sigma\mu_{\text{mean}}$, where $\sigma \geq 1$. Then, the maximum replication factor, denoted as $\rho_{\max}$, is equal to $\lceil \frac{\mu_{\max}}{M_{\text{mean}}} \rceil$, where M_{mean} stands for the average peer load and constitutes the target load where the balancing procedure should converge for all peers. Again, we assume that there are n overlay nodes that are hosted by m peers in total, with $m = g \times n$. Hence, we take that,

$$\rho_{\max} = \left\lceil \frac{\mu_{\max}}{\frac{n\mu_{\text{mean}}}{m}} \right\rceil = \left\lceil \frac{\sigma\mu_{\text{mean}}}{\frac{n}{m}\mu_{\text{mean}}} \right\rceil = \left\lceil \sigma \frac{m}{n} \right\rceil = \lceil \sigma g \rceil, \tag{9}$$

which declares the expected association between redundancy and skewness.Furthermore, our local allocation scheme that exploits replication relies exclusively on a peer's limited knowledge about the network. At each iteration of the *LBB*, we select a random peer and we redistribute all linked peers to the associated nodes. More specifically, let node u that forwards a message to w. Then, in order to distribute load among all replicas of w, u would have to deliver it to any arbitrary w's replica. Hence, u needs to know in advance all w's replicas, in order to be in position of selecting any of them. Such an approach requires a new replica of w to notify all associated nodes about its availability. Likewise, when a replica departs, it updates the routing tables of all associated nodes. Albeit no additional hops and messages are required for lookups, insertions cause an additional linear cost corresponding to updating all relevant replicas. However, these overheads can be kept low, since we can achieve our major goals with a relatively small number of replicas.

Algorithm 5 *joinLBB3:* the *LBB* algorithm that exploits *replication*

```
1: q ← getMaxLoadedLinkedPeer()
2: for all u ∈ q do
3:   p.add (u)
4: end for
```

Algorithm 6 *departLBB3:* the *LBB* algorithm that exploits *replication*

```
1: for all u ∈ p do
2:   if u.getReplFactor() > 1 then
3:     u.destroyReplica()
4:   else
5:     u.departFromOverlay()  // merged with sibling
6:   end if
7: end for
```

2.4 Discussion

The thorough study of the aforementioned balancing techniques and their intrinsic characteristics allows us to discover the tradeoffs between different methods, orthogonal to each other, and go one step further by pinpointing some emerging questions when it comes to on-line load-balancing. How much data-redundancy do we really need? How many virtual nodes should correspond to each peer? What if the load distribution changes over time? How can we dynamically adjust the number of replicas, either through creating new ones, or eliminating others, in response to network changes as peers join, leave, and fail arbitrarily? These key questions are in the core of the load-balancing problem we try to tackle in this paper. Moreover, the greatest challenge here is that these important questions have to be addressed all

at the same time by a single, flexible and robust on-line protocol. As a result, our former meticulous study provides us with the tools to conjure up an on-line balancing protocol that enables peers to congregate more than one nodes at any given time. However, these nodes are not necessarily unique, in other words, they might serve as replicas for heavily loaded virtual nodes. More importantly, this combination of techniques becomes most effective in conjunction with our parallel universes scheme.

3 The multi-bin protocol for on-line load-balancing

This section is dedicated to bringing together the aforementioned balancing methods and combine them in such a way that their effect is maximized as the overlay adapts to dynamic changes of the network topology. Thereby, we add a whole new dimension as techniques orthogonal to each other are able to work in a rather complementary fashion, operating at two different levels, affecting both the amount of traffic that the bottlenecks intake, and provide a chance for precise load adjustment through a migration-based protocol.

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. The rationale of our protocol to depend exclusively on peer load distribution, complements and preserves intact all desired overlay properties and characteristics as it does not intervene in overlay operations. In effect, data indexing, overlay operation and query processing are completely undertaken by the overlay as a whole. Our protocol operates on top of the overlay, in the entity responsible for managing, transferring, replicating and caching the congregated nodes. Besides, the load-balancing mechanism should not compromise the structural properties and thus the efficiency of the overlay network, while preserving semantic information of the data (e.g., range-partitioned networks, lexicographic ordering to enable range searches, etc.). More importantly, we upheld all restrictions concerning cardinality, as a peer may contain many nodes and a node might be replicated in many peers in the same time, for the shake of fair load distribution. Additionally, the Multi-Bin protocol pursues primarily high levels of fairness and tries to minimize the consumed bandwidth.

In principle, we augment peer joins and departures with some additional balancing steps that are shown in Algorithms 7 and 8. Even though it can be argued that some overhead is introduced, the alternatives to our migration/replication based on-line protocol demand frequent topology reorganization, introduce significant communication and maintenance costs as apart from moving around individual keys they also need to update peers' routing tables. To elaborate, consider an overlay of n virtual nodes. This systemic parameter is fixed. When a peer wants to join the network, it has to allocate at least one overlay node dispatched from some other peer. Similarly, a peer has to dislodge its nodes when it wants to depart, either by assigning them to other peers, or destroying them, if they are already replicated somewhere else in the network. Therefore, it is not possible for data to be lost in the process of balancing.

Algorithm 7 *MBjoin:* the multi-bin *join* procedure

1: **for** $j \leftarrow 1$ **to** `realitiesNr` **do**
2: $T_{1\rightarrow j} \leftarrow$ `getAverageNeighborhoodLoad(`$1 \rightarrow j$`)`
3: $G_p \leftarrow T$
4: `xph` $\leftarrow$ `createMaxPriorityPeerHeapFromOverloadedPeers(`$T_{1\rightarrow j}$`)`
5: **while not** `xph.isEmpty()` **do**
6: $q \leftarrow$ `xph.pop()`
7: $G_q \leftarrow |q$`.getLoad(`$1 \rightarrow j$`)` $- T_{1\rightarrow j}|$
8: $u \leftarrow \text{argmin}_{v\in q}|v$`.getLoad()` $- G_p|$
9: **if** $|G_p - v$`.getLoad()`$| < G_p$ **and** $|G_q - v$`.getLoad()`$| < G_q$ **then**
10: p`.add(`v, j`)`
11: **if** q`.getNodesNr() >` 1 **then**
12: q`.remove(`v`)`
13: **end if**
14: $G_p \leftarrow \left|G_p - \frac{v\text{.getLoad()}}{v\text{.getReplFactor()}}\right|$
15: $G_q \leftarrow \left|G_q - \frac{v\text{.getLoad()}}{v\text{.getReplFactor()}}\right|$
16: **else**
17: **break**
18: **end if**
19: **end while**
20: **end for**

Algorithm 8 *MBdepart:* the multi-bin *departing* procedure

1: **for** $j \leftarrow 1$ **to** `realitiesNr` **do**
2: `xnh` $\leftarrow$ `createMaxPriorityNodeHeapFromLocalNodes(`j`)`
3: `nph` $\leftarrow$ `createMinPriorityPeerHeapFromLinkedPeers(`j`)`
4: **while not** `xnh.isEmpty()` **do**
5: $u \leftarrow$ `xnh.pop()`
6: **if** u`.getReplFactor() >` 1 **then**
7: **continue**
8: **end if**
9: **if not** `nph.isEmpty()` **then**
10: $q \leftarrow$ `nph.pop()`
11: q`.add(`v, j`)`
12: p`.remove(`v, j`)`
13: `nph.push(`q`)`
14: **end if**
15: **end while**
16: **end for**

3.1 Peer joins

Algorithm 7 constitutes the procedure that is executed when a new peer p wants to join the network. Then, p has to contact an already participating peer b. Now, b initiates a balancing procedure for each reality (line 1), and as a result nodes from b's routing table are conveyed to p greedily from the heaviest to the lightest until p reaches target load $T_{1\rightarrow j}$ which stands for the average load among the associated peers (line 2), in a *best-fit* effort (line 8) from the heaviest known peer. However, this probing part of the procedure can be omitted, if peers opt for hiding additional information about their congregated nodes, their load and status, in normal traffic (piggy-backing). Additionally, a new replica of a node is created in p, when the heaviest peer known to b contains

only one node for that reality (lines 10, 11). Note that the new peer is the only peer receiving nodes while the heavy peers in b's routing table grasp the opportunity to dispatch some of their nodes (line 12).

3.2 Peer departures

We will now describe Algorithm 8, the balancing operation that a peer that departs appropriately executes. Analogously to the former procedure, the departing peer p is the sole peer making virtual nodes available by dissolving itself. In turn, peer p's heaviest node is assigned successively to the lightest peer in p's routing table. All p's nodes that are not unique are lost at p's departure by not being assigned anywhere. Naturally, the process is repeated for all realities.

3.3 Adaptive balancing

We also investigate the case where load distribution changes as no peers join or depart. We endorse a simple mechanism for treating such scenarios. Peers can naturally detect imbalances in their own neighborhood by hiding additional information about congregated nodes in normal traffic, their load and status (piggy-backing). Specifically, whenever a peer h detects that it is the heaviest peer in the neighborhood and its load exceeds c times the load of the lightest peer in the neighborhood, then peer h runs Algorithm 9, which is the adaptive version of the aforementioned algorithms that accompany peer joins and departures.

Algorithm 9 *MBadapt:* the multi-bin adaptive balancing procedure

1: **for** $j \leftarrow 1$ **to** `realitiesNr` **do**
2: $T_{1\rightarrow j} \leftarrow$ `getAverageNeighborhoodLoad`$(1 \rightarrow j)$
3: `nph` $\leftarrow$ `createMinPriorityPeerHeapFromUnderLoadedPeers`$(T_{1\rightarrow j})$
4: **while not** `nph.isEmpty()` **do**
5: $p \leftarrow$ `nph.pop()`
6: $G_p \leftarrow |T_{1\rightarrow j} - p$`.getLoad`$(1 \rightarrow j)|$
7: `xph` $\leftarrow$ `createMaxPriorityPeerHeapFromOverloadedPeers`$(T_{1\rightarrow j})$
8: **while not** `xph.isEmpty()` **do**
9: $q \leftarrow$ `xph.pop()`
10: $G_q \leftarrow |q$`.getLoad`$(1 \rightarrow j) - T_{1\rightarrow j}|$
11: $u \leftarrow \text{argmin}_{v \in q} |v$`.getLoad()` $- G_p|$
12: **if** $|G_p - v$`.getLoad()`$| < G_p$ **and** $|G_q - v$`.getLoad()`$| < G_q$ **then**
13: p`.add`(v, j)
14: **if** q`.getNodesNr() > 1` **then**
15: q`.remove`(v)
16: **end if**
17: $G_p \leftarrow \left| G_p - \frac{v\text{.getLoad()}}{v\text{.getReplFactor()}} \right|$
18: $G_q \leftarrow \left| G_q - \frac{v\text{.getLoad()}}{v\text{.getReplFactor()}} \right|$
19: **else**
20: **break**
21: **end if**
22: **end while**
23: **end while**
24: **end for**

4 Fault-tolerance

In terms of fault-tolerance, we endorse a specialized load-aware mechanism to facilitate robust operations that counteract against contingent churn failures for network stability and consistency. More specifically, we adopt a simple strategy that requires the smallest possible redundancy degree. In particular, taking advantage of the structure of our scheme, we maintain for each overlay node a *cached* replica, which can otherwise described as hidden from the overlay network, in a sense that a cached replica does not respond to any requests or issues new queries. In essence, all cached nodes are replicas that do not participate in the overlay network, but instead, they remain reserved for case of emergency, such as failures. In addition, cached notes do not contribute to any balancing efforts. Instead, a cached node's tasks are limited to monitoring the state of the overlay nodes it replicates, in order to know when it will have to take action and start serving as a surrogate node in the overlay, and of course, being kept up-to-date. Even for the case where more than one replicas correspond to a single overlay node, just one of these will serve as a back-up node, without responding to ongoing traffic. Hence, according to our scheme, peers also serve as containers for cached nodes, as well. This is a radically different approach than the prevailing techniques for similar purposes that accompany indexing infrastructures like CAN, where the owner of the data is responsible for re-inserting periodically its tuples in the index [40], and thus, leading to a communication overhead that it could have been avoided otherwise.

4.1 Cached nodes updates

In order to preserve consistency between a cached node and the original node, all changes that take place in the original node should also be applied to the associated cache as well. Hence, each overlay node is responsible for notifying its cached replica about all the changes that take place in its area of responsibility by simply forwarding certain update requests. In effect, operations that cause changes either to a node's key space or connectivity would require an additional hop to be also mirrored to the cached node.

Furthermore, a cached node has no established links, other than with the original node it replicates, and keeps updated connectivity information of the node it copies, e.g. routing table records. This kind of information is of major significance, since such an incident occurs, each one of the failed nodes' cached replica becomes automatically part of the overlay by undertaking all responsibilities of their corresponding failed node. We note that the cached replica will be activated only when *all* replicas that participate in the overlay have failed. Normally the replicas would fail one by one and only the last one will eventually trigger the cached node to be activated. Naturally, this can be easily detected and traced by catching the appropriate TCP socket exception, or by incorporating a time-out mechanism. In other words, if a cached node detects that none of the original nodes is linked anymore and all communication fails, then it takes its place in the overlay by replacing the failed node.

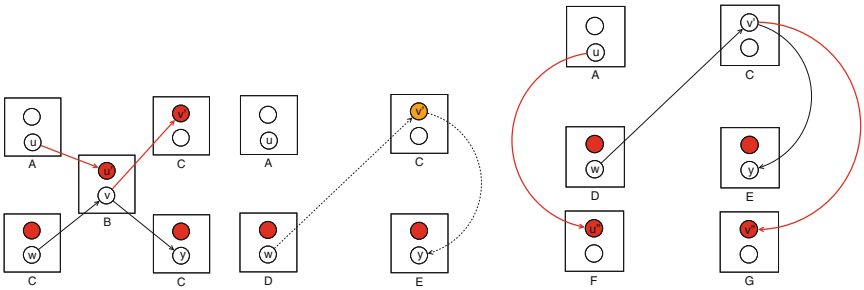
4.2 Cached nodes creation

Let node u, hosted in peer p, that wants to obtain a new cached node w. This might be the case for numerous reasons. For example, node u might have been part of a larger node v that was split to expand the overlay network. In addition, node u might also be the result of two nodes that were merged in order to form a single node. In any case, node u will issue a request for the creation of its cached replica to the least loaded peer (in terms of data-load) it knows from its routing table. Naturally, all peers provide a caching service. Additionally, any overlay node is in position of destroying its cached replica whenever it wants to, or create a new one in another host. Typical examples where we would need to destroy a cached node is when this node is either split or merged with another node, where in both cases we will need to create new replicas for the resulting node(s). Moreover, when a normal replica of a specific node is created according to our balancing protocol, then the new replica shares the same cached node with the original overlay node. If a peer p is requested to host a cached node u of a node that it already hosts, then p forwards in turn the request to the least loaded container it knows, since u has no cached node. On the other hand, when peer p that already contains a cached replica of u is assigned responsibility of u itself, for example due to a balancing operation, it destroys the local cache, fulfills the request and finally forwards in turn the cache construction request to the least loaded peer it knows. Otherwise, when a peer is requested to construct a node that it already hosts or its cached replica, then that request is simply ignored.

4.3 Counteracting failures

In this section, we examine the sequence of operations and actions to be undertaken upon the detection of a crashed node. Initially, a cached node w is linked to the node u it replicates. If that communication fails, then w might try to re-establish it, in order to assert node u's failure. If the outcome of this effort proves to be vain, then node w will become part of the overlay by replacing u and activating the links with the nodes in its routing table and sending notifications to the nodes that u was linked with before it crashed, in order to inform them about the change and provide service. Hence, the surrogate node w is now ready to participate in the overlay network. Nevertheless, node w will first try to leave properly the overlay according to the departure protocol, for example by being merged with its sibling. If this is not possible, then we keep node w as is. In that case, we will need to create a cached node for node w in another peer, as we described in Sect. 4.2.

In Fig. 3, we visualize this mechanism with a typical example for reconstructing lost overlay nodes and cached replicas due to a crashed peer. Figure 3a depicts all cached replicas with red fill. Then, as illustrated in Fig. 3b, peer B fails, and as a result, overlay node v and cached node u' that replicates node u are lost. The dotted edges represent the procedure taking place for surrogate node v' to establish communication with the overlay nodes whose records finds in its routing table, and



(a) Peer B with node v and cached node u' fails. **(b)** Surrogate node v' links with w and y. **(c)** Nodes u and v' create cached nodes u'' and v'', respectively.

Fig. 3 Handling a typical peer failure example appropriately

consequently, become itself part of the overlay by replacing lost node v. Figure 3c depicts peers F and G and their newly created cached replicas u'' and v'' of overlay nodes' u and v' respectively, that will also serve as surrogates in case of churn failure.

Furthermore, if the cached node that failed served more than one overlay nodes, then each of them will try to create a new cached node of itself. However, only the first one will succeed in doing so. The rest of the requests will be ignored essentially, without any problem or inconsistency as all replicas will try to create a cached replica into the same peer due to its light load, as we discussed in Sect. 4.2.

5 Experimental evaluation

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

5.1 Setting

Our experimental evaluation consists of two major parts. In the first part, different allocations for rigorous balancing techniques are compared, whilst the latter part consists of dynamic simulations. We are especially interested in how our paradigm performs compared to other types of allocation. Therefore, we compare our methods with a greedy centralized allocation for each technique and as a means to investigate the potential of this approach. More specifically, as an optimality measure, 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 the latter part of this section we evaluate our dynamic protocol. More specifically, we study the course of specific metrics over dynamic simulations in order to draw useful conclusions about effectiveness and performance. The virtual overlays used in our experiments consist of $10K$ nodes in all cases. We discern three main consecutive simulation stages:

- Growing stage, a new peer is added to the network in each cycle. We initialize a network of 1,250 peers and add one peer at each cycle, using a randomly selected bootstrap peer, until the network consists of $80K$ peers.
- Steady stage takes the same number of cycles, each introducing a peer join followed by a departure. We select arbitrarily where peers join, and departing peers, as well.
- Shrinking stage, a random peer at each cycle is removed until the network reaches its initial size.

In all experiments, we evaluate our balancing protocols in conjunction with the MIDAS-RDF framework [43,44] that serves as a distributed RDF repository, and the PGrid-Z [14] overlay, which incorporates a Z-curve into P-Grid to support multidimensional range search. We assume that all peers have identical network capabilities. We also assume that all requests, regardless how they are processed (whether are answered locally or simply forwarded), are modeled with the same service demand. These conventions were made in order to ensure that our experiments will not be affected by the distributions of those parameters. We also assume that the load on a virtual node is stable (or can otherwise predicted) over the timescale it takes for the load balancing algorithm to operate. Hence, the load of a peer can be calculated by summing the loads of the comprised nodes, and thus, a peer's load is also stable. Concerning baselines, we place against our own scheme a competitor that leverages topology reorganization principles, for instance [20]. Recall that this family of protocols includes two main types of operations to mitigate load imbalances, namely *neighbor adjust*, to transfer data between adjacent nodes, and *reorder*, to hand over data to neighbor and split load of some other distant overloaded node. More importantly, this strategy does not require any additional redundancy.

5.2 Metrics

We evaluate our methods by various metrics, mostly to contention related. Latency is the maximum distance in terms of hops from the initial peer to any peer reached during search. Albeit, maximum process throughput, $\Lambda_{\max}$, 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 task-load fairness. Specifically, we consider a peer's task-load as the number of queries it receives, and thereby, must either forward or process locally. In particular, we use Jain's fairness index [28] to measure task-skew, as it is applicable to any number of peers, it is independent of scale, and it is continuous in [0, 1], where a totally fair system should have a fairness

of 1. This fairness index is expressed in the form, $FI = (\sum_{i=1}^{N} x_i)^2/(n \sum_{i=1}^{N} x_i^2)$, where x_i stands for peer's i load, with $x_i \geq 0 \; \forall i$.

Moreover, an important cost arises from redundancy and enhanced availability, which is imposed by all techniques with the exception of virtual nodes. We consider successful a method that is capable of exploiting available data-redundancy in order 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 important metric, as a peer that contains many nodes has to deal with high data load and maintenance cost. Last but not least, we also examine the communication cost in terms of exchanged data tuples, and withal, required bandwidth.

5.3 Data sets

For the purposes of this work, we make use of spatial data sets describing roads and rivers of Greece, taken from the R-tree portal [2], using a piecewise linear approximation, and a data set containing the locations of towns and villages in Greece. The final data set consist of 100K tuples. Each query set consists of 10K range queries, as this type of search is considered important for applications and is also very resource consuming. In addition, we define three cluster-centers corresponding to the largest cities of Greece: Athens, Thessaloniki and Patras; and all clusters follow complex normal distributions. Skewed query sets consist of queries generated from a cluster chosen with probability proportional to the population of each town. The query construction process for all query sets is totally synthetic. We pick one point from the data set, we displace it by a random amount and we create a square query around this point until it evaluates to 50 tuples. Moreover, we created synthetic data sets of variable dimensionality to study the impact of dimensionality. Query sets consist of range queries and query centers are randomly chosen as query bounds are expanded over all dimensions starting from that point until query evaluates to 50 points.

Furthermore, we evaluate range queries with a real data set of approximately 11.2M triples. The DBLP data set [1] is available in XML format and contains a large number of bibliographic descriptions on major computer science journals and proceedings, more than half a million articles and several thousand links to home pages of computer scientists. For our evaluation, we use an RDF converted data set from XML of the Proximity DBLP database. Specifically, the data in this data set were derived from a snapshot of the bibliography in 2006. In fact, only seven distinct predicates appear in the outlandish percentage of 72% of our triples. We evaluate MIDAS-RDF with the most frequent types of queries that are used by inference methods that implement the W3C RDFS entailment rules [3]. In particular, with *rangeXZ* we denote query sets consisted of queries with fixed predicate and varying subject and object. With *rangeZ* query sets of range queries with fixed subject and predicate and varying object. Unlike the previous data sets, RDF triple pattern queries, like the ones examined here, are non-selective queries. As a result, more peers are detained as part of the answer process, and the need for a rigorous and effective balancing scheme is even more crucial. These query sets consist of approximately 50K queries.

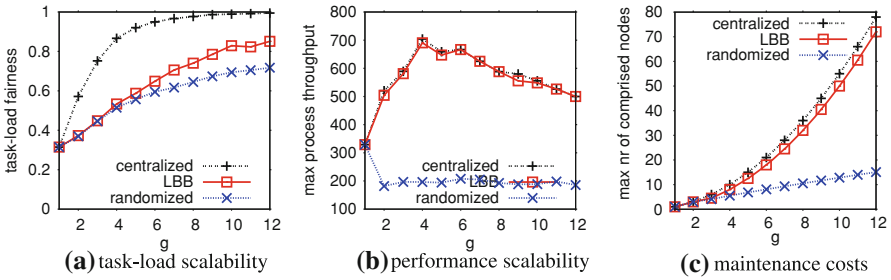


Fig. 4 Virtual nodes schemes for spatial workloads and network size = 10K

5.4 Results

In this section, we study and evaluate our methods for various values of the system parameters, depending on the method studied on each occasion, and we draw useful conclusions from their behavior of the selected metrics. For the static part of our experiments, we examine the effect of the key parameter g which corresponds to either (i) the number of virtual overlay nodes to the overall peers for the virtual nodes scheme, or (ii) the number of peers to the number of the overlay nodes for the replication balancing scheme, or (iii) the number of realities used. On the whole, there is a significant benefit from using our methods instead of using the straightforward naive approach, denoted as *randomized*. Figures 4, 5, and 6 illustrates the effectiveness of our methods compared to allocations based on global overlay knowledge (centralized), denoted as *centralized*, which perform only slightly better compared to our schemes that rely exclusively on peers' partial knowledge about the overlay.

Regarding the *virtual nodes* technique, maximum process throughput shows a concave behavior in Fig. 4b, as predicted in Sect. 2.1. An explanation to this behavior can be found to the 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 even higher g values. Therefore, this method is unsuitable for very large networks, as performance would deteriorate instead of improve. Thus, given an overlay of specific size, there are certain values of g that ameliorate $\Lambda_{\max}$. In effect, 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. In addition, congregating numerous nodes increases maintenance costs. We note that apart from keeping multiple routing tables, peers are burdened with maintenance tasks, such as detecting failures. Moreover, latency and precision are affected due to the method's direct interaction with the overlay size. In particular, 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, and thus, when peers join the network by selecting a key with equal probability (data balanced), they tend to populate densely those areas as well. In practice, 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

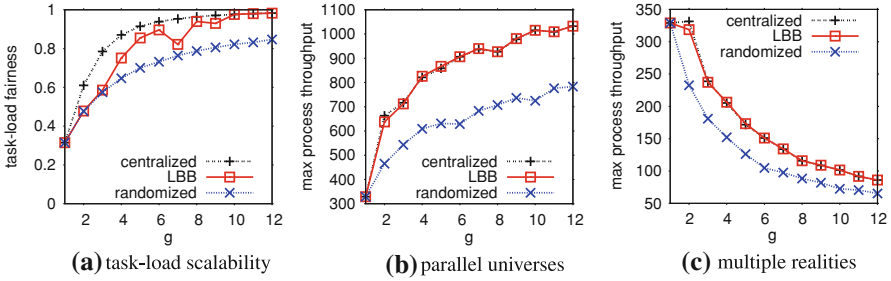


Fig. 5 Redundant overlay schemes for spatial workloads and network size = 10K

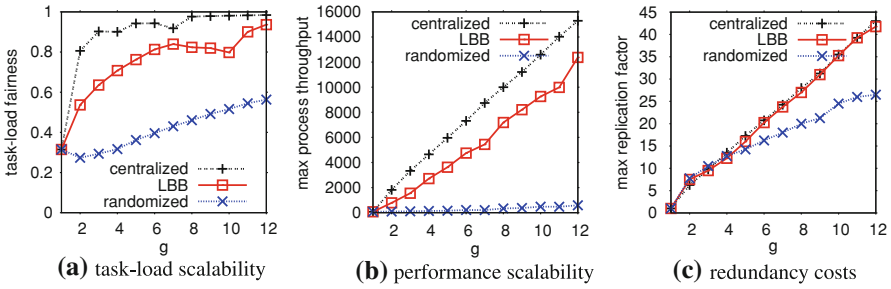


Fig. 6 Replication schemes for spatial workloads and overlay size = 10K

method imparts the overlay with max process throughput, precision and latency from larger overlays with respect to g. Concerning the randomized assignment of nodes to peers, we adopt a naive straightforward strategy for each node to be assigned to any peer with equal probability. For the centralized allocation scheme, the heaviest node is assigned greedily to the globally lightest peer at the time. Clearly, as Fig. 4b depicts, our approach outperforms randomized assignments as it provides 3.5 times better $\Lambda_{\max}$. More importantly, centralized allocations are less that 5 % more efficient than our scheme for all configurations and g values.

In Fig. 5b, our *parallel universes* technique outperforms the *multiple realities* technique in terms of maximum process throughput as it grows with the number of realities g, while *multiple realities* max process throughput deteriorates in Fig. 5c. Moreover, we compared our paradigm with different allocation methods. Regarding the straightforward randomized allocation, we assume for each reality that nodes are assigned to peers arbitrarily. For the centralized allocation we consider again a method that is run for each reality requiring global knowledge. More specifically, we assign a heaviest node to the lightest peer with respect to a peer's summing load in all realities preceding the one being examined. In addition, task-load fairness index ameliorates (Fig. 5a) regardless dimensionality or skewness.

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 addition, the invariant latency can be explained as routing takes place along the overlay and not

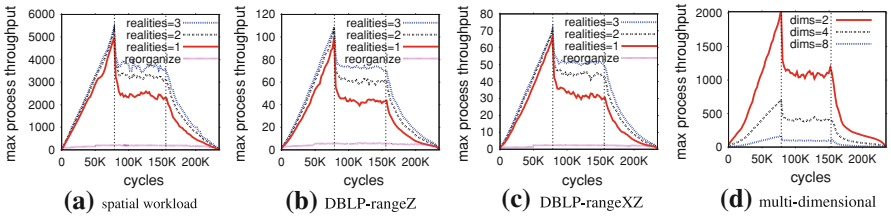


Fig. 7 Maximum process throughput for various workloads

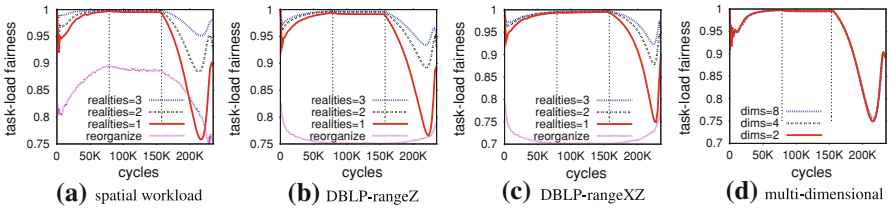


Fig. 8 Task-load fairness for various workloads

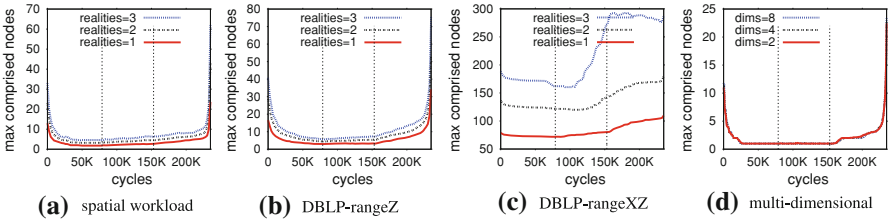


Fig. 9 Maximum comprised nodes in a single peer for various workloads

the actual network. Concerning competitor schemes, for the straightforward randomized allocation, each host replicates a randomly selected node with equal probability. The centralized allocation is an algorithm that requires global knowledge about all nodes and their loads, and each time it replicates the heaviest node to an available host. Figure 6b shows that there are immense benefits of using our method instead of the corresponding randomized allocation. It also reveals that this is the most effective technique in terms of max process throughput, as it selectively replicates only hotspots. More notably, we repeated our experiments for different overlay and network sizes to receive very similar results.

For the rest of this section, we focus on the results from our dynamic simulations where the network is subjected to extreme changes regarding its size for all phases (Figs. 7, 8, 9, 10, 11). We note that latency depends on the fixed size of the virtual network topology, regardless the number of actual hosts that have joined the network. Specifically, for both the distributed indexing schemes that we evaluate here, namely PGrid-Z and MIDAS-RDF, it is logarithmic with respect to the overlay size. In Fig. 7, maximum process throughput shows for all configurations an abrupt slope at the end of the growing phase.

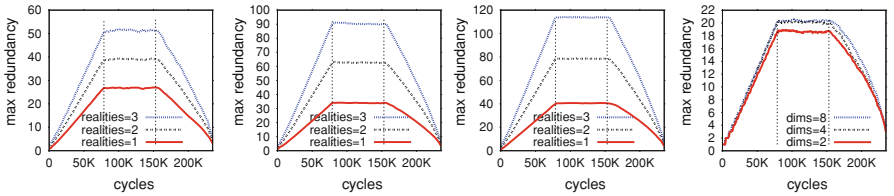


Fig. 10 Replication factor of the most replicated node for various workloads

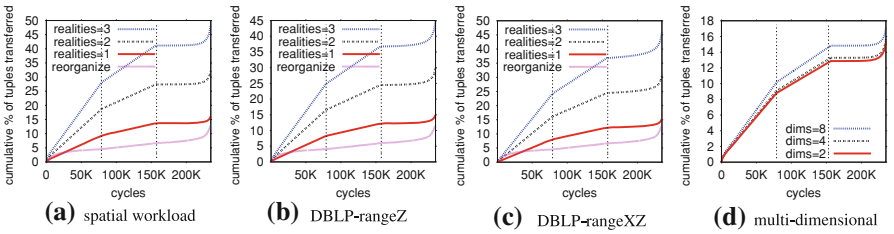


Fig. 11 Aggregated communication cost for various workloads

Presumably, when a sequence of departures takes place, the nodes that are more likely to be removed are the most replicated and heavy nodes. Besides, at this point there are 8 times more peers than virtual nodes. Consequently, it is plausible for the replicas of those nodes to become affected the most, as changes to the number of their replicas affects max process throughput drastically. In particular, for the DBLP RDF workloads, in Fig. 7b and c, $\Lambda_{\max}$ diminishes dramatically due to the nature of the non-selective triple pattern queries and the arising immense task-load. Our method outperforms the reorganization protocol by some orders of magnitude in terms of $\Lambda_{\max}$ at all cases, due to the fact that our protocol exploits available data-redundancy and organizes replicas appropriately.

On the other hand, task-load fairness shows convex behavior in Fig. 8 during growing and shrinking stage. More specifically, fairness takes its lowest values when the number of peers equals the number of virtual nodes, as the benefits of the virtual nodes technique before it, and replication afterwards, improve fairness significantly. We note that the network is initialized with an optimal allocation of overlay nodes to peers (ratio 8:1), and therefore, it starts from high-levels of task-load fairness for all configurations, in an effort to investigate whether they can be retained as peers join and leave the network. Quite remarkably, increased dimensionality in Fig. 8d fails to affect task-load fairness at all! Additionally, the increasing the number of coordinate realities used has a beneficial impact on load fairness and $\Lambda_{\max}$ for the spatial data sets, Figs. 7a and 8a, and the DBLP RDF workloads, Figs. 7b, c, and 8b, c, as well. However, the reorganization protocol fails to achieve high fairness levels.

In Fig. 9, the maximum number of comprised nodes in a single peer shows a different behavior for each stage of our simulations. For all settings, it decreases during the growing stage as more and more peers join the network and allocate overlay nodes, while the number of (unique) overlay nodes remains fixed, as our balancing scheme

dictates. However, especially for the least selective workload we examine in Fig. 9c, this metric increases during the last part of the steady phase until the end of the simulation. Specifically, congregating numerous nodes increases maintenance costs. Apart from keeping multiple routing tables, peers are burdened with maintenance tasks, such as detecting failures. Moreover, another collateral of our scheme, the most replicated overlay node quantifies the redundancy costs, in Fig. 10, that increase linearly with network size and the number of realities. In both Figs. 9 and 10 we show the maximum value that the presented metrics take for each configuration. Albeit not presented here, the average case is sensible improved. If we study meticulously both figures, we will realize that due to the flexibility some parts of the network use the virtual nodes technique, while some very "popular" nodes of the overlay are replicated by many hosts in the network at the same time. In fact, the greater the redundancy, the less traffic bottlenecks intake, and thereby, imbalances are blunted as overloaded peers are alleviated and no peer becomes unduly loaded. On the other hand, there is a one-to-one correspondence for the reorganization protocol which maps peers to virtual nodes, and thus, minimizing both cost aspects.

Last but not least, bandwidth is probably the most expensive resource for a distributed system. Figure 11 depicts, in a cumulative fashion, the recorded percentage of the data set that changed ownership between any pair of peers. In particular, a tuple is counted every time it is being transferred from one peer to another, due to a peer's arrival or departure. By the end of the growing stage, tuples that equal approximately to 8 times the data set change hands for all settings, as the network increases from 1,250 peers to 80K peers. We also observe that the growing and shrinking stages exhibit asymmetric behaviors, linear and convex, respectively. Apparently, steepest slopes appear when the overlay nodes outnumber the peers for all metrics, during the first part of the growing stage and the latter part of the shrinking stage. Bandwidth requirements are significantly less for the reorganization protocol. However, toward the end of all simulations they seem to be close for a single reality used, mainly due to the fact that eliminating replicas by simply discarding them when a peer departs, is less costly than transferring individual multi-dimensional keys between neighboring peers. Finally, the communication costs increase linearly with the number of realities, as we expected.

6 Related work

In this section we summarize hitherto most conspicuous load-balancing efforts, denoting similarities and discrepancies with our work. In particular, we group them according to the leveraged approach.

6.1 Balls-in-bins models

This section introduces essential preliminaries and establishes the necessary background regarding balls-in-bins games. In the standard balls-in-bins model for balanced allocations, m balls are placed sequentially into n bins. Each ball probes the loads of d uniform-random bins and is placed in the least loaded bin. For this model, maximum

load refers to the maximum number of balls in any bin at the end of the process. For $d > 1$ the maximum load is $\frac{\log \log n}{\log d} + O(1)$ whp [11].

More general bin-selection processes are studied in [31] by allowing each ball to choose a random pair of bins from only a subset of possible pairs. In [23] correlation when choosing bins is introduced, as each ball i belongs in an associated random bin-set B_i. It is required that $|B_i| \in \Omega(\log n)$ and bins are included in B_i with approximately the same probability. Godfrey argues in [23] that the correlations in bin choices can be such that a ball which picks a "hot-spot" bin shares the same d alternatives as all other $\frac{\log n}{\log \log n}$ balls that picked the same hot-spot. He shows that the distribution of balls in bins is approximately dominated by a process of placing slightly more than m balls into uniform-random least-loaded bins. On the down-side, balls are identical (equal loads) according to the aforementioned schemes. Since computing an optimal reassignment that minimizes maximum peer utilization is NP-hard, [22] suggests a greedy algorithm to find an approximate solution.

6.2 Consistent hashing

In this section we describe a family of hashing and caching protocols for distributed networks that are used to decrease or eliminate the occurrence of hotspots. The conventional balancing paradigm for DHTs consists of randomized hashing functions that are supposed to distribute data load among peers in a uniform fashion. Probably the simplest remedy to load-balancing a peer-to-peer system is the "power of two" [15,34] approach, whereby an item is stored at the least loaded of d alternatives, given by associating a d hash values with a key. Under these methods, d hash functions are used to pick candidate bins for each item to be inserted. The load of each bin is compared and the item is inserted into the least loaded bin. Hence, if there are n bins, n items and $d \geq 2$ hash functions, the maximum load of any bin is $\frac{\log \log n}{\log d} + O(1)$ whp. Therefore, to search for an item, one applies the hash functions and examines each bin to locate the item. Nevertheless, traversing multiple paths introduces Messages overhead analogously to d. Most importantly, this solution is limited for lookups, whereas we are interested in overlay networks that preserve proximity and order among indexed keys, and can therefore support complex operations (e.g., Mercury [12], MAAN [16]), such as range queries. 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 [45]. Besides, if semantics are associated with the item IDs, the imbalance factor becomes arbitrarily bad, since IDs are no longer uniformly distributed.

Karger and Rhul propose a family of caching protocols to decrease the occurrence of hot-spots. *Address-space balancing* [30] improves consistent hashing [29] 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. Then, each node has a fixed set of possible positions, and it chooses exactly one of those potential nodes to become active. Given a set of active nodes, each potential node "spans" a certain range of addresses between itself and the suc-

ceeding active node on the address ring. Each real node has activated the potential node that spans the minimal address. Nodes for which that ideal state condition is not satisfied, activate the potential nodes for which it is satisfied and the process repeats. Ramabhadran et al. adapt in [38] a trie-based structure (prefix hash tree, PHT) to a DHT. Although their structure is generic, range queries are highly inefficient since locality is not preserved.

6.3 Restructuring schemes

In this section we discuss the most conspicuous algorithms and protocols that are used to mitigate imbalances by reorganizing the overlay and the areas of the key-space that peers are responsible for. *The threshold algorithm* [20] 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 is that a node attempts to shed 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. In fact, the greater the dimensionality, the harder it is to redefine boundaries between two neighboring peers, in response to equalizing their loads.

Item balancing [30] targets at the distribution of items to nodes. The protocol is executed periodically by a peer i, probing randomly the load l_j of another peer j and triggered when one peer has ϵ times greater load than the other. If the two peers are neighbors ($i = j + 1$) and $l_i > l_j$, then j increases its address so that the $\frac{l_i - l_j}{2}$ items with the lowest addresses in i's interval get reassigned from i to j. If i and j are not neighbors and $l_{j+1} > l_i$, then set $i := j + 1$ and go to the previous case. Otherwise, node j moves between nodes $i - 1$ and i to capture half of node i's items, and now j's items are handled by its former successor. Similar approach followed in [32].

Reference [13] constitutes an effort to balance through keeping the length of intervals assigned to the peers vary at most by a constant factor. In short, their algorithm minimizes the length of the longest interval, but also taking care that no interval is too short. However, such an approach addresses data-skew only tacitly, while task-skew over data is not considered at all. MURK [21] is a quite noteworthy effort for a scheme that pursues an even distribution of data-keys to peers in CAN [40].

On the other hand, Mercury [12] supports explicit load balancing using random sampling; nodes are grouped into routing hubs, each responsible for various attributes. Similarly, [35] views a peer-to-peer system as comprising clusters of peers and presents techniques for both intra-cluster and inter-cluster load-balancing.

6.4 Migration-based schemes

Rao et al. introduce in [39] the notion of the *virtual server*, a single node of the underlying DHT, unlike the physical host being responsible for more than one virtual servers. The key advantage of splitting load into virtual servers is that we can move a virtual server from any host to any other. The load L_i of peer i is equal to the sum of the loads of its virtual servers. It is assumed that every node has a target load T_i

and therefore is considered to be heavy if $L_i > T_i$ and light otherwise. The fundamental operation performed is transferring a virtual server from heavy nodes to light nodes.

Nevertheless, [39] does not discuss dynamic scenarios where the number of peers is not stable, especially for cases where the network grows in a degree that peers outnumber virtual servers. Most importantly, the topology of the virtual servers should be able to adapt to dynamic changes of the actual network accordingly.

An extension of this concept is presented in [22] that reduces the distributed load balancing problem to a centralized problem at each directory. Each directory has an ID known to all nodes and is stored at the node responsible for that ID. Upon joining the system, a node u reports to a random directory the loads $u_1, \ldots, u_m$ of its comprised virtual servers and its capacity c_u. Every T seconds, a directory computes a schedule of virtual server transfers among those nodes with the goal of reducing their maximum utilization to a parametrized periodic threshold. After completing a set of transfers scheduled by a directory, a node chooses a new random directory and the process repeats. Nevertheless, maintaining multiple globally known directories imposes high maintenance cost in order to keep them up-to-date, and withal, load to nodes is introduced that is not considered. Moreover, it can be argued that simply transferring popular virtual servers from peer to peer only transfers the problem. Besides, related research in web proxies has testified the need of replication [46] as replication offers important advantages, such as fault tolerance, high availability [35]. In another pertinent work, Skip-bins [9] adheres to the same principle by adopting a pairing strategy in which heavily-loaded machines can be dealt with by migrating elements to their lightly-loaded neighbors. Specifically, the authors propose a mechanism based on statistical sampling and by targeting the local adjustments made by individual nodes to respond to changes in load to avoid massive migratory stampedes.

6.5 Replication-based schemes

HotRoD [36] deals with balancing through the use of a locality-preserving hashing function and a tunable data replication mechanism that allows trading off replication costs for fair load distribution. Each peer in HotRoD keeps track of the number of times it was accessed during a time interval, and the average low and high bounds of the ranges of the queries it processed. A peer is overloaded when its access count exceeds the upper limit of its resource capacity. An arc of peers is "hot" when at least one of these peers is overloaded. HotRoD replicates and rotates "hot" arcs of peers over the identifier space. Thereby, a HotRoD instance consists of a regular DHT ring and a number of overlapping rings where values are addressed using a multi-rotation hash function. However, HotRoD cannot be combined with overlays that rely on a structured pattern other than a ring, for example a torus as in CAN [40]. Nevertheless, relying exclusively on data replication implies effective management in terms of storage and updates cost, even when an imbalance could be resolved by simply transferring a popular "arc" to a less utilized peer with available resources. LAR [24] is a replication framework that relies on server load measurements to precisely choose

replication points for Chord [42]. They also augment the routing process with specific routing information, "hints" as they call them, that effectively shortcut the original routing and direct queries towards new replicas.

Another balancing effort that relies on replication is proposed in [7] that exploits the structure of P-Grid [4–6,8,17], a distributed prefix-tree structure that supports range queries. An alternative approach that relies on caching is discussed in [18] by the same research group. However, caching data-keys proportional to their access frequency might also lead to high task-load imbalances when routing to any of these caches randomly. A radically different approach is presented in [19] where a load-aware routing strategy is proposed that requires modifications of the routing mechanisms and a piggy-backing protocol to estimate traffic.

7 Conclusions

To recapitulate, we concurrently addressed the issues of replication/migration-based load-balancing and efficient query processing in structured peer-to-peer systems with a robust load-balancing protocol. Our proposed protocol can be implemented on top of any structured peer-to-peer system, regardless indexing scheme or dimensionality degree. In addition, our work is complemented with extensive experiments using large real and synthetic multi-dimensional data sets, which we evaluated over the PGrid-Z and the MIDAS-RDF frameworks. Specifically, we showed that our scheme achieves high levels of fairness and outperforms topology reorganization protocols by some orders of magnitude, a gain which comes at a cost of maintenance and replication cost that we manage efficiently, and thus, being compensating in many ways. Moreover, we make the most out of the structural properties of our scheme by augmenting our work with an effective load-aware fault-tolerance mechanism to ensure resiliency and counteract against contingent churn failures.

An indispensable part of this paper studies static rudimentary balancing techniques which we improve and incorporate them in our dynamic protocol. More specifically, the virtual nodes technique improves performance for certain configurations due to its combined effects, without imposing additional redundancy. However, this method accumulates data load from all comprised nodes. The multiple realities technique is considered appropriate exclusively for lookup queries as range queries impair performance dramatically, whereas our parallel universes scheme 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. Finally, our experimental evaluation demonstrates that our proposed techniques enhance performance, scalability, and also achieve high levels of fairness.

References

1. The dblp data-set. http://dblp.uni-trier.de/xml
2. The r-tree portal. http://www.rtreeportal.org
3. W3c rdfs rules of entailment. http://www.w3.org/TR/rdf-mt/#rules

4. Aberer K (2001) P-grid: a self-organizing access structure for p2p information systems. In: CoopIS, pp 179–194
5. Aberer K (2002) Scalable data access in peer-to-peer systems using unbalanced search trees. In: WDAS, pp 107–120
6. Aberer K, Datta A, Hauswirth M (2004) Efficient, self-contained handling of identity in peer-to-peer systems. IEEE TKDE, 16
7. Aberer K, Datta A, Hauswirth M (2005) Multifaceted simultaneous load balancing in DHT-based p2p systems: a new game with old balls and bins. In: Babaoglu Ö, Jelasity M, Montresor A, Fetzer C, Leonardi S, van Moorsel APA, van Steen M (eds) Self-star properties in complex information systems, pp 373–391
8. Aberer K, Datta A, Hauswirth M (2005) P-grid: dynamics of self-organizing processes in structured peer-to-peer systems. In: Steinmetz R, Wehrle K (eds) Peer-to-peer systems and applications, pp 137–153
9. Aspnes J, Kirsch J, Krishnamurthy A (2004) Load balancing and locality in range-queriable data structures. In: PODC, pp 115–124
10. Avidor A, Azar Y, Sgall J (2001) Ancient and new algorithms for load balancing in the l_p norm. Algorithmica 29(3):422–441
11. Azar Y, Broder AZ, Karlin AR, Upfal E (1994) Balanced allocations. SIAM J Comp 29(1):180–200
12. Bharambe AR, Agrawal M, Seshan S (2004) Mercury: supporting scalable multi-attribute range queries. In: SIGCOMM, pp 353–366
13. Bienkowski M, Korzeniowski M, auf der Heide FM (2005) Dynamic load balancing in distributed hash tables. In: IPTPS, pp 217–225
14. Blanas S, Samoladas V (2009) Contention-based performance evaluation of multidimensional range search in peer-to-peer networks. Future Gener Comp Syst 25(1):100–108
15. Byers JW, Considine J, Mitzenmacher M (2003) Simple load balancing for distributed hash tables. In: IPTPS, pp 80–87
16. Cai M, Frank MR, Chen J, Szekely PA (2004) Maan: a multi-attribute addressable network for grid information services. J Grid Comp 2(1):3–14
17. Datta A, Hauswirth M, John R, Schmidt R, Aberer K (2005) Range queries in trie-structured overlays. In: P2P computing, pp 57–66
18. Datta A, Nejdl W, Aberer K (2006) Optimal caching for first-order query load-balancing in decentralized index structures. In: DBISP2P, pp 331–342
19. Datta A, Schmidt R, Aberer K (2007) Query-load balancing in structured overlays. In: CCGRID, pp 453–460
20. Ganesan P, Bawa M, Garcia-molina H (2004) Online balancing of range-partitioned data with applications to peer-to-peer systems. In: VLDB, pp 444–455
21. Ganesan P, Yang B, Garcia-Molina H (2004) One torus to rule them all: Multidimensional queries in p2p systems. In: WebDB, pp 19–24
22. Godfrey B, Lakshminarayanan K, Surana S, Karp RM, Stoica I (2004) Load balancing in dynamic structured p2p systems. In: INFOCOM
23. Godfrey PB (2008) Balls and bins with structure: balanced allocations on hypergraphs. In: SODA '08, pp 511–517
24. Gopalakrishnan V, Silaghi BD, Bhattacharjee B, Keleher PJ (2004) Adaptive replication in peer-to-peer systems. In: ICDCS, pp 360–369
25. Graham RL (1969) Bounds on multiprocessing timing anomalies. SIAM J Appl Math 17(2):416–429
26. Jagadish HV, Ooi BC, Vu QH (2005) Baton: a balanced tree structure for peer-to-peer networks. In: VLDB, pp 661–672
27. Jagadish HV, Ooi BC, Vu QH, Zhang R, Zhou A (2006) Vbi-tree: a peer-to-peer framework for supporting multi-dimensional indexing schemes. In: ICDE, p 34
28. Jain R, Chiu D, Hawe W (1984) A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. In: DEC research report TR-301
29. Karger D, Lehman E, Leighton T, Panigrahy R, Levine M, Lewin D (1997) Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: ACM symposium on theory of computers, pp 654–663
30. Karger DR (2004) Simple efficient load balancing algorithms for peer-to-peer systems. In: ACM SPAA, pp 36–43
31. Kenthapadi K, Panigrahy R (2006) Balanced allocation on graphs. In: SODA '06, pp 434–443

32. Konstantinou I, Tsoumakos D, Koziris N (2011) Fast and cost-effective online load-balancing in distributed range-queriable systems. IEEE Trans Parallel Distrib Syst 22(8):1350–1364
33. Maymounkov P, Mazières D (2002) Kademlia: a peer-to-peer information system based on the xor metric. In: IPTPS, pp 53–65
34. Mitzenmacher M (2001) The power of two choices in randomized load balancing. IEEE Trans Parallel Distrib Syst 12(10):1094–1104
35. Mondal A, Goda K, Kitsuregawa M (2003) Effective load-balancing via migration and replication in spatial grids. In: DEXA, pp 202–211
36. Pitoura T, Ntarmos N, Triantafillou P (2006) Replication, load balancing and efficient range query processing in dhts. In: EDBT, pp 131–148
37. Raab M, Steger A (1998) Balls into bin—a simple and tight analysis. In: RANDOM, pp 159–170
38. Ramabhadran S, Ratnasamy S, Hellerstein JM, Shenker S (2004) Brief announcement: prefix hash tree. In: PODC, p 368
39. Rao A, Lakshminarayanan K, Surana S, Karp RM, Stoica I (2003) Load balancing in structured p2p systems. In: IPTPS, pp 68–79
40. Ratnasamy S, Francis P, Handley M, Karp R, Schenker S (2001) A scalable content-addressable network. In: SIGCOMM '01, pp 161–172
41. Rowstron AIT, Druschel P (2001) Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware, pp 329–350
42. Stoica I, Morris R, Liben-Nowell D, Karger D, Kaashoek F, Dabek F, Balakrishnan H (2003) Chord a scalable p2p lookup protocol for internet applications. IEEE/ACM Trans Netw 11(1):17–32
43. Tsatsanifos G, Sacharidis D, Sellis TK (2011) Midas: multi-attribute indexing for distributed architecture systems. In: SSTD, pp 168–185
44. Tsatsanifos G, Sacharidis D, Sellis TK (2011) On enhancing scalability for distributed rdf/s stores. In: EDBT, pp 141–152
45. Wang X, Loguinov D (2007) Load-balancing performance of consistent hashing: asymptotic analysis of random node join. IEEE/ACM Trans Netw 15(4):892–905
46. Wu K-L, Yu PS (2003) Replication for load balancing and hot-spot relief on proxy web caches with hash routing. Distrib Parallel Databases 13(2):203–220
47. Zhao B, Kubiatowicz J, Joseph AD (2004) Tapestry: a resilient global-scale overlay for service deployment. IEEE J Sel Areas Comm 22(1):41–53