Policies for Efficient Data Replication in P2P Systems

João Paiva, Luís Rodrigues
INESC-ID, Instituto Superior Técnico, Universidade Técnica
{joao.paiva,ler}@ist.utl.pt

***Abstract*—This paper addresses the problem of maintaining replicated data in large scale P2P systems. Although this topic has been extensively studied in the literature, to maintain replicated data in this setting, *in an efficient manner*, still remains a significant challenge. This paper proposes novel policies to address this problem and evaluates their performance against different criteria, such as monitoring costs, data transfer costs, and load unbalance costs. We show that one of these new policies significantly outperforms previous work. Interestingly, this policy is based on a somehow counter-intuitive approach, that uses less reliable nodes to store the most accessed data items. The insights to derive this policy were obtained from an in depth analysis of existing solutions, that is also captured in the paper.**

Keywords*—*Fault-tolerance, Data replication, P2P, Bandwidth, Load balancing, Group-based DHTs.

I. Introduction

Any large scale system is subject to failures of individual nodes. In P2P systems, where nodes are typically COTS (Components Of the Shelf), instead of highly reliable servers operated under controlled conditions in datacenters, the occurrence of failures is far from negligible[1]. Furthermore, some P2P systems may be subject to the churn effect that results from end users connecting and disconnecting volunteer resources at high pace[2]. All this phenomena may cause data loss, unless some form of data replication strategy is used.

Given the relevance of implementing data replication in P2P systems, this topic has been extensively studied in the literature. Surprisingly, despite all the results achieved so far, to maintain replicated data in this setting in an efficient manner still remains a significant challenge. In fact, in a talk at Middleware 2011, Drushel and Rowstron have identified data replication as one of the problems for which no satisfactory solution had been proposed yet.

What makes data replication particularly challenging is that maintaining the replication degree incurs several costs, and the goal of reducing the costs in one dimension of the problem typically conflicts with the goal of reducing the costs in the remaining dimensions. Namely, in this paper we concentrate on the following 3 different relevant costs metrics of a data replication scheme:

- *Monitoring costs:* the costs associated with monitoring existing replicas to assess if they are still live, such that new replicas are spawned timely to replace failed replicas. This is required to preserve the desired replication degree and ensure that data is not lost.

- *Data transfer costs:* the costs associated with the creation of new replicas in the system. Blake and Rodrigues [3] have shown that data transfer costs account for a significant part of the bandwidth consumption in the system supporting replication and that these costs effectively limit the amount of data that can be stored.

- *Load unbalance costs:* some replication policies may result in an unbalanced replica distribution among existing nodes. Load unbalancing has a detrimental effect on the operation of the system, since some nodes will be overloaded while the capacity of other nodes will be underused.

As hinted above, to optimize all these costs simultaneously may be impossible. For instance, some replication policies favor the placement of replicas on nodes that are known to be more reliable. This allows for saving in data transfer costs (since the creation of new replicas becomes less frequent) at the cost of introducing load unbalancing (the more reliable replicas become overloaded). Furthermore, how these tradeoffs are tackled by a specific algorithm is often hard to infer, as proposers of a given replication scheme typically evaluate their solution using only a subset, if not only one, of the metrics above, neglecting the impact of the proposed policy on the other dimensions of the problem. In face of these observations, the current paper makes the following contributions:

i) It provides a comparative study of the most relevant data replication policies that have been proposed in the literature, highlighting the tradeoffs they implement when considering the different costs involved.

ii) More importantly, it proposes a number of novel data replication policies, that have not been previously experimented in the literature.

iii) It shows that some of the novel policies outperform previous strategies. In particular, one of these policies when compared with other competing solutions, achieves large bandwidth savings, offers good load balancing, and has no negative impact on the monitoring costs. The best policy is based on a strategy that may appear counter-intuitive at first sight: it uses the less reliable nodes to store the most accessed data items. The rationale for the success of this strategy will be clear later in the text.

The remaining of this paper is structured as follows. To better motivate our work, in Section II, we start by highlighting the tradeoffs implemented by previous data replication policies, by analyzing their performance in face of experimental data. Then, in Section III we survey existing approaches and also propose a number of novel policies. A performance model that allows to assess the effectiveness of each approach is proposed in Section IV. With the help of this model, the different polices are experimentally evaluated and compared in Section V. Finally, Section VI concludes the paper.

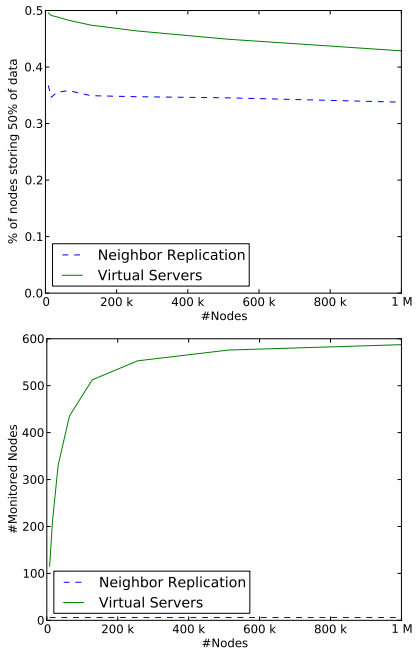


Fig. 1: Load balancing vs monitoring costs tradeoff

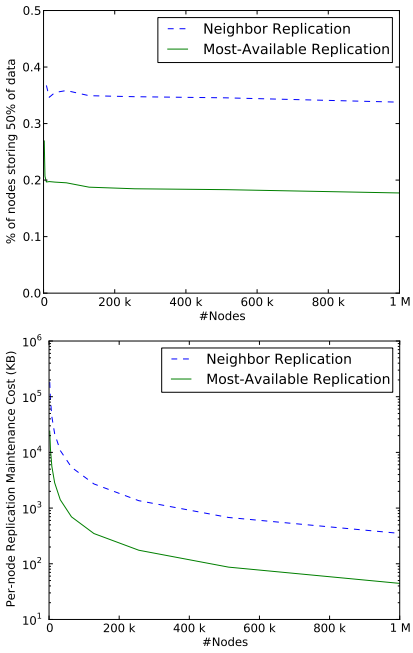


Fig. 2: Load balancing vs data transfer costs tradeoffs

II. Tradeoffs in Data Replication

The reader may find surprising that, given the huge body of research in data replication in the context of P2P systems[4], [5], [6], [7], this is still an open topic. However, to the best of our knowledge, no studies have been published addressing the costs of data replication on its multiple dimensions. The work of [3] has focused on the data transfer costs and its impact on the scalability of the system. The work [8] has addressed data transfer costs and load unbalance, while the work [9] focused mostly on monitoring costs. As a result, it remains particularly difficult to understand the tradeoffs involved in the different replication policies that have been proposed in the literature.

To motivate our work, we illustrate our point using 3 well-known replication policies:

- *Neighbor Replication[6], [4]:* This is a widely used policy that has been implemented on many P2P systems based on DHTs, such as Chord [6] or Pastry [4]. A primary replica is selected for each item (typically using consistent hashing) and replicas are maintained in the neighbors of the primary replica in the overlay.

- *Neighbor replication with virtual servers[7], [10]:* Similar to the policy above except that each node joins the overlay with multiple identities.

- *Most-available node replication[11]:* This policy consists in placing replicas in the nodes that are less likely to fail or to leave the overlay.

Figures 1 and 2 show the comparative performance of these policies according to different metrics (we will postpone a detailed description of the experimental setting to Section V, where we do a more detailed analysis of these and other replication policies). Figure 1 illustrates the tradeoff between the load unbalancing costs and the monitoring costs of the first two policies above. The introduction of virtual servers augments the number of (virtual) nodes in the overlay and, therefore, promotes a good load balancing among the replicas. These advantages are well documented in the literature[7], [10]. On the other hand, each node has to keep track of a different set of neighbors for each of its identities. Therefore, the improvements in load balancing come at the cost of a proportional increase in the monitoring costs. Figure 2 illustrates the tradeoff between load unbalacing costs and the data transfer costs when comparing Neighbor Replication *vs* Most-available node replication. By selecting the most reliable nodes to store replicas, fewer items are affected by failures and fewer replicas need to be respawned. This introduces significant savings in the data transfer costs, as reported in [11], [8]. Unfortunately, these gains come at the cost of unbalancing the load in the system. In fact, we have observed that the unbalance can be 2000 times worse than other solution (see Sec V).

The examples presented before provide an insight on the problems addressed in the current paper. Namely, the paper addresses the following questions: Do other previously proposed policies implement similar tradeoffs? Are there replication policies that can reduce the costs in a given metric without significantly increasing the costs in another metric? Are there unexplored policies that can implement more advantageous tradeoffs? Is it possible to define a framework that helps in comparing the performance of different policies?

To answer these questions, we will first classify the main replication policies that have been proposed in the literature, then propose some novel policies, and finally provide a detailed comparative analysis of the different alternatives.

III. A Catalog of Previous and New Policies

In this section we start by introducing a classification that helps in comparing the policies for data replication according to the their principles of operation. Subsequently, using this

classification, we list the most relevant approaches that have been proposed in the literature. Finally, we propose some novel policies that, to the best of our knowledge, have not been presented before.

A. Policy Classification

We first distinguish *oblivious* from *informed* policies. Oblivious policies do not take into consideration the properties or state of each peer and consider only topology properties of the overlay. For instance, neighbor replication, introduced before, fits in this category. On the contrary, informed policies collect information about each individual node, such as the expected availability, or the current load, to make decisions about data placement. For instance, the Most-available policy, fits in this category.

Another important dimension that can be used to classify replications policies considers the machinery required to locate replicas. Here we distinguish two opposing strategies: *consistent hashing* and *directory-based* lookups. When consistent hashing is used, replica location is derived by the hashing function. As a result, the policy has little control on the data placement but, on the other hand, replicas can be located in a very efficient manner. Directory-based approaches can place replicas in arbitrary locations but require a lookup to the directory to find the replica location; this may involve one or more round-trips in the network.

Finally, we can distinguish policies according to the requirements they impose on the underlying overlays. In this paper we considered exclusively replication policies for structured overlays that implement some form of DHT. However, besides traditional DHTs such Chord [6] or Pastry [4], we also consider overlays that use virtual servers[7] and logical groups. Virtual servers were already introduced in Section III-B. Logical groups are discussed next.

A number of recently proposed overlays take a different approach to replication [5], [12], [13], [14]. These overlays create self-contained replication groups of nodes which act as single nodes in the DHT. Routing is performed at the group level and not at the physical node level, allowing the system to fine-tune the replication sets. So, even though they are based on consistent hashing, these overlays allow to decouple the management of replication from management of the overlay topology. In group-based DHTs, each node is a logical entity, materialized by a replica group of variable size. Contrary to classical DHTs where each node has a pre-determined location in the network (depending on its identifier), in these approaches nodes can join any existing replica group. All members of each group coordinate to act as a single node in a higher layer, defining a DHT. Since replication is decoupled from the DHT layer, the replication degree of individual groups can vary (with a configurable interval) without affecting the DHT's structure. As a result, consistent hashing is used to place data items into groups, and data is then replicated among all group members.

In Table I we consolidate the different criteria together. In the next paragraphs we discuss how they can be combined to build different strategies:

- *Strategies based on consistent hashing:* Strategies based on consistent hashing rely the node (or group) identifiers and on randomization to place replicas. In this case, virtual servers only offer better randomization, because the number of nodes ids is larger. If the DHT is not-group based, there are little opportunities to improve the operation of the network, other than carefully selecting the node identifiers, to compensate for load unbalancing. In the case of group-based DHTs it is also possible to change the size of the groups, as nodes join and leave, or even by migrating nodes from one group to the other.

- *Directory-based strategies:* Oblivious strategies that use directories place replicas randomly but can maintain the replicas in the same nodes regardless of their position in the DHT, to avoid moving replicas. Informed strategies can do better, by placing replicas taking into consideration the properties of nodes. However, directories bring no advantages when group-based DHTs are used, because groups have no intrinsic characteristics: the characteristics of each group can be tuned as will by the policy as described before.

B. Previous Replication Policies

We now identify and describe the most relevant policies that have been previously proposed in the literature.

1) Strategies based on consistent hashing:

a) Oblivious Strategies: We consider the following three oblivious strategies that rely on consistent hashing: neighbor replication, multi-publication, and neighbor replication with virtual servers.

Neighbor Replication (NR): Neighbor replication[6], [4] (also known as leaf set replication) keeps copies of each key-value pair in the r neighbors of the node responsible for the key (named the *key owner*). A significant advantage of Neighbor Replication is that it allows to keep a tight control on the replication degree: when the neighbors change, the key owner may trigger the creation of new replicas to ensure that the replication degree does not fall below a target threshold. As demonstrated in [3], this has a high replica maintenance cost, since replicas must be created and destroyed with every change in the network. However, given the fact that each node replicates on a fixed set of neighbors, which is independent of how many data items are stored in the system, this solution has low monitoring costs.

Multi-Publication (MP): Multi-Publication[15], [16] stores r replicas of each data item in different and deterministically correlated positions of the DHT. Then, some mutual monitoring scheme needs to be implemented to detect the departure/failure of a replica and subsequently restore the replication degree. The main advantage of Multi-Publication is that it offers very good load balancing properties, as multiple queries may be diverted to different regions of the DHT. On the other hand, monitoring becomes extremely expensive, because it needs to use DHT routing and a node may be forced to monitor a different set of nodes for each object that it stores. In the worst case, a node that is the owner of M objects, each replicated in r other locations, has to periodically monitor $M \times r$ different nodes.

Neighbor Replication with Virtual Servers (NR+VS): Consistent-hashing solutions such as neighbor replication are not immune to load balancing problems; nodes, keys, or (more likely) the load imposed by different keys may not

TABLE I: Design Space of Replication Policies for P2P Systems

	consistent hashing			directory based		
	plain	**virtual servers**	**groups**	**plain**	**virtual servers**	**groups**
oblivious	baseline	achieve better randomization	manage group size to avoid churn at the logical level	avoid migrating replicas		no advantage
informed	change node ids to distribute keys		change groups ids and place nodes in the right groups	place replicas in better nodes		

be uniformly distributed in the address space. As a result, some nodes will be required to maintain (and answer queries for) many items while others may be relatively offloaded. A common technique to circumvent this problem is to use virtual servers [7], [10]. In this technique, each physical node joins the system using multiple identities; each identity represents a virtual node maintained by that server. However, the efficiency of this mechanism depends on how many virtual servers each node can handle, as a larger number of such servers will lead to an improved load distribution. On the other hand, having a larger number of virtual servers involves each node maintaining more routing information and monitoring more overlay neighbors, which may impose an excessive overhead. Also, this strategy amplifies the effect of churn (a phenomenon characterized by the rapid changing in system membership [2]), as the departure of a single node causes the simultaneous failure of multiple virtual nodes.

b) Informed Strategies: To the best of our knowledge, the only informed strategy that does not require the use of a directory has been proposed in the context of group-based DHTs in [5] (the name has been given by us, as the original Scatter paper provides no names to the proposed policy).

Resilient Load-Balancing (R-LB): This policy uses the underlying overlay mechanisms to address resiliency and load-balancing. Resiliency is tackled by having nodes join groups with fewest members and by merging a small group with its successor. Furthermore, the Lazy Node Load-Balancing attempts to balance the load of each individual node, by making sure that when a group is split into two, each new group will process a load proportional to its size. To implement this policy, the overlay mechanisms are configured as follows: new nodes join the group with the highest *per-node* load; group identifiers (and therefore key assignment) remains unchanged until a group needs to be split or merged; when the group splits, keys are divided such that the per-node load of each resulting group is balanced (i.e, the identifier of the new group is chosen such that it owns a portion of the load proportional to its number of members).

2) Directory-based strategies:

a) Oblivious Strategies: We consider the following oblivious strategy that requires the use of a directory.

RelaxDHT: This policy, presented in [17], is a natural extension of neighbor replication. Initially, replicas are created in the r closest neighbor of the item owner. However, unlike neighbor replication, as new nodes join the neighbor set of the item owner, the constraint that replicas should be the closest nodes to the item owner is relaxed. Hence, replicas are not moved as new nodes join, and may drift away from the item owner. To lower the monitoring costs, the work in [17] imposes a limit on how farther away from the item owner replicas may drift. The main intuition behind this work is that by relaxing the topological constraints, one can achieve lower replica maintenance costs. However, the fact that nodes which are "older" in the system tend to store more replicas, leads to an imbalance in load.

b) Informed Strategies: We consider the following informed strategies: Most-available and regularity-based.

Most-available: The work in [11] presents a policy which places data in the nodes predicted to be most available in the future. Similarly to the RelaxDHT policy, data is never relocated from a node as long as it is available. Hence, even though this technique can achieve particularly low replication maintenance costs, it leads to high monitoring costs.

Regularity-based: This policy, introduced in [8], takes advantage of the fact that nodes may exhibit connection regularity. This means nodes connect to the system on regular patterns. Hence, the system can form groups of nodes which with a given probability will always be online to replicate the data. Unlike other works, this policy assumes nodes can keep persistent state between joining and leaving the system. The regularity-based policy biases replication towards a set of nodes (the most regular ones), which impacts negatively load balancing in the system.

C. Novel Policies

As mentioned in the previous section, Group-based DHTs open very interesting avenues to design novel policies for data replication in P2P systems and the work of Scatter has only explored the surface of these possibilities (in fact, in their paper, the authors of [5] recognize the development of such policies as an intriguing direction for future work). In this paper we follow that path by proposing a number of viable alternatives to the R-LB policy introduced in [5].

1) Oblivious Strategies: We consider the following three oblivious strategies for group-based DHTs:

Random: This policy achieves load balancing through randomization. It works by letting nodes generate a random identifier and join the group responsible for that identifier. Similarly, when a group becomes too large, it divides, creating a new group with a random identifier which joins at a random location. When a node becomes too small, it disbands and all its members join in random locations. The goal of this policy is to serve as a baseline that can be used to assess the relative merit of other strategies for group-based topologies.

Supersize-me: This policy consists in tuning the overlay management mechanisms to keep groups much larger than the average replication degree which the policy is configured with. In this way, the amount of redundancy is increased but the likelihood that a group collapses due to the lack of replicas becomes very small. This may avoid some unnecessary data transfers in the system.

TABLE II: Policy Map

		primary performance target			
		none	**monitoring**	**load balancing**	**bandwidth**
oblivious	Plain		Neighbor Replication (NR)	Multi-Publication	RelaxDHT
	VServers			Neighbor Replication + Virtual Servers (NR+VS)	
	Groups				Supersize-me
informed	Plain				Most-available, Regularity-based
	Groups	Random	Avoid-Surplus	R-LB	Preemptive replacement
			Hotter-On-Ephemeral		

Avoid Surplus: This policy consists of the opposite of the "Supersize-me" policy above. The goal is to keep each group as close as possible to minimum replication degree, by having nodes join the largest groups such that those divide more frequently. This reduces the amount of redundancy in the system but creates the potential for better load balancing (more groups will exist in the overlay) and for reducing the monitoring costs (which is directly related to the group size).

2) Informed Strategies: We consider the following two oblivious strategies for group-based DHTs:

Preemptive Replacement:. The rationale for this policy is to adapt ideas that have been proposed for classical DHTs, such as the "Most-available" policy introduced in Section III-B to group-based DHTs. The key idea is to rely on estimates of node-reliability to make new nodes join groups where existing nodes are most likely to fail, as a preemptive measure to avoid the group to become to small and forced to execute a merge.

Hotter-On-Ephemeral (HonE): This policy aims to place the most used items (i.e. those which represent the highest load) on less reliable nodes. The key observation behind this policy is that the per-group distribution of keys of R-LB reduces load unbalance but ignores which nodes compose which groups. This insight, combined with the fact that the groups which store the top most used keys will store less keys to achieve a balanced load, drives the design of Hotter-On-Ephemeral. So, Hotter-On-Ephemeral places the less reliable nodes in the groups which in R-LB store less keys, i.e., the groups that store the most accessed keys in order to maintain the good control over Load-Unbalance and the good monitoring costs of R-LB, while considerably reducing its bandwidth usage, since most joins will be performed in groups with fewer keys. In order to make sure that the groups remain stable, this policy also allows groups composed of mostly unreliable nodes to grow larger than the remaining ones, similarly to "Supersize-Me".

D. Summary of All Policies

Table II provides a summary of all policies described in this section. The lines represent different techniques and the columns the main performance criteria that is aimed by the policy. Note that the Hotter-On-Ephemeral aims at addressing the three performance criteria in a holistic manner.

IV. A Performance Model To Compare Policies

We now introduce two metrics that can be used to compare the performance of different policies: the *message overhead* and the *unbalance ratio*. These two metrics are defined as a ratio between the performance of an actual system using a given policy against the performance of an abstract idealized system (referred as the baseline system).

A. System Parameters

Our metrics rely on the following system parameters: *Number of Nodes (N)*, The average number of nodes in the system; *Number of Keys (K)*, The total number of keys stored in the system; *Replication Degree (R)*, The desired replication degree (each key should have R replicas); *Key Size (d)*, The amount of stored data associated with each key; *Churn Frequency (c)*, The number of joins and leaves per unit of time; *Monitoring Period (m)*, The number of units of time between two consecutive monitoring procedures; *Unit Monitoring Cost (UC_m)*, The cost of checking if another node is alive (typically, at least one "I'm alive" exchange); *Unit Transfer Cost*, ($UC_t(d)$): The cost of transferring one key from one node to another (depends on the data size d); *Load (L)*, The total number of requests the system has to satisfy per unit of time.

B. Baseline System

Using the parameters above, we define the following performance metric for an abstract idealized baseline system. The idealized system is completely homogeneous: the load is uniformly distributed among keys, such that if all nodes store exactly the same number of keys, then all nodes are subject to the same load. Furthermore, the probability of a node leaving the system is the same for every node and, when a node fails, its load and objects are scattered uniformly across all other nodes, such that these values are preserved. Also, in this system, nodes monitor the bare minimum number of neighbors for maintaining the target replication degree (i.e. R).

Baseline Average Number of Keys per Node (BK_{avg}): The average number of keys per node is $BK_{avg} = K \cdot R/N$.

Baseline Monitoring Cost (BC_m): We consider that a node should be responsible for at least one key, and therefore has to monitor at least R other nodes. Therefore $BC_m = R \cdot UC_m$.

Baseline Join/Leave Cost ($BC_{join\_leave}(d)$): We denote the baseline join/leave cost as the cost of transferring BK_{avg} keys to a node, i.e., $BC_{join\_leave} = BK_{avg} \cdot UC_t(d)$.

Baseline Load Balancing (BLB): In the idealized system all nodes have the same load. We define the baseline load balancing as the fraction of system nodes that, together, satisfy half of the system load in this setting, which, by definition, is 0.5 (i.e., half of the nodes are required to serve half of the system load).

C. Actual System

The same metrics can be defined for a concrete system, using a particular policy. Namely:

Actual Average Number of Keys per Node: (AK_{avg}) The actual average number of keys per node that results from applying a given policy.

Actual Monitoring Cost: (AC_m) The actual (average) monitoring cost that results from applying a given policy. This depends on how many nodes each peer must keep monitoring.

Actual Join/Leave Cost ($AC_{join\_leave}(d)$) : The actual (average) join/leave cost that results from applying a given policy. This depends on how many data items need to be transferred upon each join and upon each leave.

Actual Load Balancing (ALB): The fraction of system nodes that, together, satisfy half of the system load. Let L_i be the number of request per unit of time served by node i. Let $\mathcal{H} \subset \mathcal{N}$ be the smallest subset of the system nodes such that $\sum_{i \in \mathcal{H}} L_i = L/2$. Then $ALB = \frac{|\mathcal{H}|}{N}$.

D. Policy Efficiency

Finally, using the metrics above, we define the two criteria to measure the efficiency of a given policy. These metrics are defined as ratios between the performance of the actual system and the performance of the idealized system. Namely:

Message Overhead ($O_{message}(d)$): We define the message overhead of a given policy as the ratio between the actual and baseline values for the sum of the monitoring and join costs during a monitoring period:

$$O_{message}(d) = \frac{N \cdot AC_m + c \cdot m \cdot AC_{join\_leave}(d)}{N \cdot BC_m + c \cdot m \cdot BC_{join\_leave}(d)}$$

Unbalance Ratio (UR): We define the unbalance ratio of a given policy as the ratio between the baseline load balancing and the actual load balancing:

$$UR = \frac{BLB}{ALB}$$

In the next section, we will evaluate the different policies using the two metrics above. Note that, for both policy efficiency metrics, the higher the value, the less efficient is the policy with regard to the idealized baseline system.

V. Evaluation

In this section we present experimental results that compare the performance of the replication policies presented in the previous section. To evaluate the policies, we have performed extensive simulations using the Peersim simulator [18] cycle-based engine, running the different policies against similar workloads in large-scale settings, as described below.

A. Simulation Settings

In order to simulate real working conditions of a large-scale deployment of a key-value store, we have used a real-world trace of connections and disconnections in a peer-to-peer network [1]. The trace represents the activity of over 14 million unique users, of which we considered 1 million random users. We used one second of the trace as unit of time.

TABLE III: Evaluation Parameters and Results for Evaluated Policies.

System Parameter Values		$O_{message}(1KB)$	$O_{message}(1MB)$	$O_{message}(1GB)$	UR
N	15434				
K	100000				
R	6				
c	3.3838				
m	60				
UC_m	84B				
$UC_t(d)$	84B $+d$				
L	1931660				
Neighbor (NR)		1.00	1.00	1.00	1774.1
NR+VS		46.73	1.27	1.18	114.5
Multi-Publication (MP)		17.09	1.21	1.18	1.5
RelaxDHT		0.11	0.21	0.21	2365.5
Most-available		0.07	0.13	0.13	2365.5
R-LB		0.71	0.52	0.52	1.1
Random		0.71	0.60	0.60	19.5
Avoid Surplus		0.76	0.41	0.41	308.5
Supersize-me		1.07	0.79	0.79	1.1
Preemptive		0.50	0.30	0.30	25.7
HonE		0.61	0.28	0.28	1.1

We populated the system with 100.000 key-value pairs with a load following a Zipf distribution with $\alpha = 2.5$. For all group-based DHTs, we have configured the DHT with min_group_size and max_group_size 4 and 8, respectively. We experimentally determined this configuration to yield an average virtual node size of 6 for policies (except, for obvious reasons, for Supersize-me and Avoid Surplus), which according to [5] and [14] is enough to prevent data loss in most scenarios. For fairness, we also configured the remaining policies with replication degree 6. For the policies that allow the group size to surpass max_group_size, we used $max\_group\_size = 16$. The virtual servers configuration used 100 virtual identities. Table III presents the remaining values for all parameters of the evaluation.

B. Results

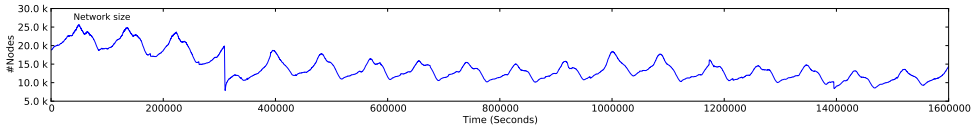
The results from our experiments, namely the values for the message overhead (as a fuction of different item sizes) and the unbalance ratio, as defined in the previous section, are depicted in Table III. The table presents the values that have been experimentally measured using the simulation for all the policies. We remind the reader that, the higher the value, the poorly the policy performs.

We start by discussing the results concerning the message overhead of the different policies. These results are summarized in the following list of observations:

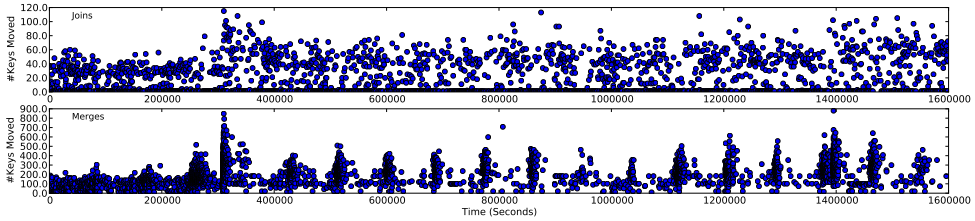
- As expected, Neighbour Replication (NR) has 1.0 for the bandwidth, since it has the exact same behaviour of the idealized baseline system.

- Virtual Servers (VS) and Multi-Publication (MP) have a high impact on bandwidth, especially when d is small, since the monitoring messages represent a good part of the cost. When using VS, each node monitors on average 587 neighbors, while when using MP, each node monitors on average 209 neighbors. In fact, however, should we consider a larger set of keys, MP would achieve an even worse result, since for each key a node owns, it potentially monitors R different neighbors.

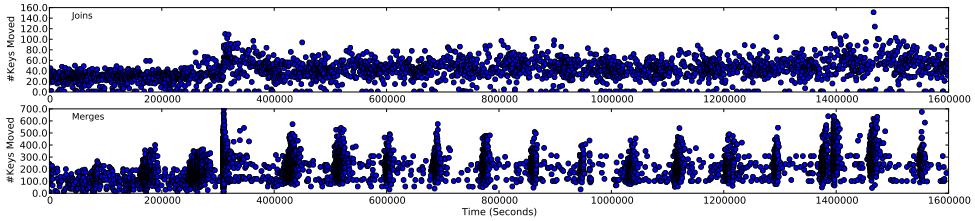
- RelaxDHT and Most-available achieve the lowest bandwidth usage of all. This is expected, given that these strategies make



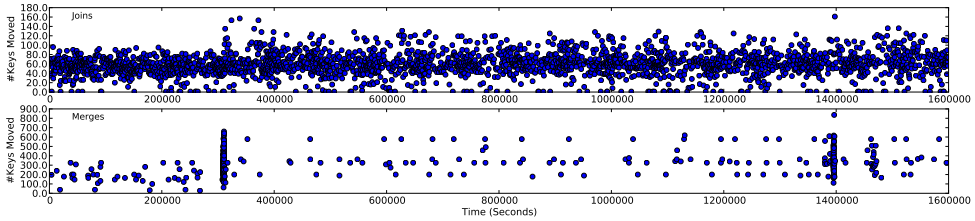
(a) Network size over time.



(b) Behaviour of HonE over time.



(c) Behaviour of R-LB over time.



(d) Behaviour of Supersize-me over time.

Fig. 3: Behaviour of policies over time

sure that the nodes that fail the most do now own many keys. Interestingly, due to the fact that many nodes do not store any keys, these nodes are not required to monitor other neighbours, and for small values of d, these three strategies in fact achieve better $MessageOverhead$ results.

- Most of the group-based policies result in a bandwidth improvement, especially when d is larger. For small values of d, group-based policies have a similar degree of replication as the baseline, and thus achieve a similar monitoring cost. For large values of d, group-based policies benefit from the fact that they support variable replication degree. Thus, while the baseline solution (and NR) must move $BC_{join\_leave}(d)$ objects for each join and leave, group-based policies can allow the replication degree to lower after a node leaves, and hence avoid moving $BC_{join\_leave}(d)$ objects for about half of c. This fact is clearly observable for R-LB, which achieves 50% less message overhead for the larger values of d.

- The group-based solution that obtains the best bandwidth usage is HonE, which despite creating groups slightly larger than preemptive (noticeable by the higher message overhead for low d), can improve over the latter solution for larger key sizes. This is due to HonE actively forcing nodes to fail on groups with a smaller number of keys, while preemptive is only making sure that groups remain stable and failed nodes are most of the time replaced with new nodes.

- From the group-based policies, the Supersize-me policy achieves the worse results. Even though it does lead to a considerably smaller number of merges, in fact it forces nodes to transfer more data with each node join, since the groups being larger also causes each individual group to store more keys on average. On the other end, HonE benefits from the best bandwidth usage, due to avoiding group merges and directing ephemeral nodes to groups with few keys. In fact, for larger values of d, HonE can achieve a result comparable to that of the best overall solution (Most-available), with a message overhead little more than $2\times$ worse.

We now analyze the results when considering the load unbalance that is caused by each policy. Again, we summarize our findings in a list of observations:

- All policies which are not designed to explicitly handle load, present results considerably worse than those of the baseline. The most flagrant cases are for the RelaxDHT and Most-available policies, which are not only oblivious of the load each key represents, but also allow a large percentage of nodes

to store 0 keys (up to 10% for Most-available).

- Virtual servers (VS) and Multi-Publication (MP) can obtain a low load unbalance ratio (despite the very negative effect on bandwidth/monitoring). MP achieves the best results, since it makes sure that the probability that two nodes store the same keys is low, while when using VS, it is common that two virtual servers own the same data. Hence, should one VS have a load above or bellow average, this effect is amplified due to R other VS replicating its data.

- The Avoid Surplus policy results in a poor load unbalance ratio, as groups have no extra capacity to absorb failures, forcing merges that difficult the task of balancing the load.

- The Supersize-me policy is able to achieve a low load unbalance ratio. This is due to, in practice, its join and division operations being similar to those of R-LB, except for the fact that the size of the groups is allowed to fluctuate up to larger values. As a result, only the message overhead costs are negativelly impacted.

- As expected, Preemptive and Random achieve poor results for load unbalance due to the lack of load balancing concerns. On the other hand, R-LB and HonE achieve similar results, both very close to the baseline, due to being designed with load balancing in mind.

C. Detail

Figures 3a, 3b, 3c and 3d present detailed views of how many keys are moved in the system as result of joins and merges, as well as the network size, as time progresses in the simulation. For clarity, we present only 0.1% randomly sampled points for join operations. The remaining points (for the network size and merges) are presented raw. For reasons of space, we present detailed views only for the most interesting policies, HonE, R-LB and Supersize-me. We highlight the following aspects of these solutions:

- Globally, it is clear that all group-based approaches have the common trend of creating merges when the network size decreases. In fact, the size of the network follows a diurnal-nocturnal pattern (there are more active nodes during the day), and during the transition to the night, the network shrinks and more merges are created. This is particularly clear around the 300.000 seconds mark, where a strong descent on the number of nodes causes not only an increase in the number of merges but also on the number of keys moved by each merge.

- In respect to the network joins, HonE shows consistently lower values for the number of keys moved than R-LB and Supersize-me, which means that more frequently, when nodes join, they actually receive very few keys. Furthermore, since HonE makes an active effort to maintain the network relatively stable, it is also able to achieve fewer merges than R-LB (about 30% less on average), and most merges end up moving as many keys as in R-LB.

- Supersize-me leads to considerably fewer merges. In fact, except for situations when the network size drops abruptly, Supersize-me is able to mostly avoid them. However, it is also observable that the increased size of the groups has a negative effect on the number of keys moved as a consequence of joins.

VI. Conclusions

Although a significant amount of literature exist on data replication policies for large-scale distributed P2P systems, to design algorithms that can provide both low monitoring costs, low data transfer costs, and good load balancing remains challenging. In this paper we have made a comprehensive analysis of the literature, witch provides interesting insights on the tradeoffs involved. Based on these insights we were able to propose a novel policy, based on a principle that can be counter-intuitive at first: to put the most accessed data items on the less reliable nodes. We have shown that this policy, named "Hotter-On-Ephemeral", significantly outperforms previous work. This work opens an interesting research avenue, that consists in designing new overlay topologies that can maximize the principle unveiled by our research.

Acknowledgments: This work was partially supported by Fundação para a Ciência e Tecnologia (FCT) via the INESC-ID multi-annual funding through the PIDDAC Program fund grant, under projects PEst-OE/EEI/LA0021/2013, CMU-PT/ELE/0030/2009 and PTDC/EEI-SCR/2776/2012.

References

[1] S. Blond, F. Fessant, and E. Merrer, "Finding good partners in availability-aware P2P networks," in *SSS*, 2009.

[2] D. Wu, Y. Tian, and K.-W. Ng, "Analytical study on improving DHT lookup performance under churn," in *IEEE P2P*, 2006.

[3] C. Blake and R. Rodrigues, "High availability, scalable storage, dynamic peer networks: Pick two," in *USENIX HotOS*, 2003.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *ACM/IFIP/USENIX Middleware*, 2001.

[5] L. Glendenning, I. Beschastnikh, A. Krishnamurthy, and T. Anderson, "Scalable consistency in scatter," in *ACM SOSP*, 2011.

[6] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM*, 2001.

[7] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *ACM SOSP*, 2001.

[8] A. Pace, V. Quema, and V. Schiavoni, "Exploiting node connection regularity for DHT replication," in *IEEE SRDS*, 2011.

[9] A. Ghodsi, L. Alima, and S. Haridi, "Symmetric replication for structured peer-to-peer systems," in *DBISP2P*, 2007.

[10] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems," in *IEEE INFOCOM*, 2004.

[11] J. Mickens and B. Noble, "Exploiting availability prediction in distributed systems," in *USENIX NSDI*, 2006.

[12] H. Abu-Libdeh, H. Geng, and R. van Renesse, "Elastic replication for scalable consistent service," in *ACM SOSP (extended abstract)*, 2011.

[13] T. Shafaat, B. Ahmad, and S. Haridi, "Id-replication for structured peer-to-peer systems," in *Euro-Par*, 2012.

[14] J. Paiva, J. L. ao, and L. Rodrigues, "Rollerchain: a dht for efficient replication." in *IEEE NCA*, 2013.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *ACM SIGCOMM*, 2001.

[16] P. Knezevic, A. Wombacher, and T. Risse, "Enabling high data availability in a DHT," in *IEEE GLOBE*, 2005.

[17] S. Legtchenko, S. Monnet, P. Sens, and G. Muller, "Churn-resilient replication strategy for peer-to-peer distributed hash-tables," in *SSS*, 2009.

[18] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "The Peersim simulator," `http://peersim.sf.net`.