

Aurelia: Building Locality-Preserving Overlay Network over Heterogeneous P2P Environments

Di Wu, Ye Tian, and Kam-Wing Ng

Department of Computer Science & Engineering,
The Chinese University of Hong Kong,
Shatin, N.T., Hong Kong
{dwu, ytian, kwng}@cse.cuhk.edu.hk

Abstract. Traditional DHT-based overlays destroy data locality and make it hard to support complex queries (e.g., range query, similarity query, etc) in P2P systems. Additionally, the node heterogeneity is widely ignored in most existing overlay designs. In this paper, we propose a locality-preserving overlay design, called Aurelia, which can adapt to the heterogeneous P2P environment and exploit node heterogeneity to realize efficient routing and robust indexing. Aurelia preserves the data locality by abandoning the use of hashing for data placement and each peer is responsible for a continuous range of value. In Aurelia, the routing table size and index range are proportional to the node capacity, and multicasting is adopted for scalable routing table maintenance.

1 Introduction

In decentralized P2P systems, data locality is an important property to enable efficient complex queries, for the cost of performing a query is often proportional to the number of peers that need to process the query. Without data locality, a simple range query will require visiting a number of peers. To efficiently support complex queries (e.g., range query, similarity query, etc) in P2P systems, the overlay should preserve the data locality to guarantee data with similar values be stored in nearby nodes.

What is more, in real P2P file-sharing environments, previous measurements [1] have identified the tremendous heterogeneity among P2P nodes. The peers exhibit great diversity in their capacity (e.g., CPU processing, disk speed, bandwidth, etc) and behaviors (e.g., lifetime, altruism). However, most of current overlay protocols simply regard them as homogeneous during design. We believe it is an unreasonable assumption and also will cause the loss of efficiency. Obviously, it is a kind of waste for powerful nodes to maintain the same size of routing table as the weak nodes.

Based on the above design rationale, in this paper, we propose a locality-preserving overlay design, called Aurelia ¹, which can exploit the node heterogeneity in bandwidth, storage and processing capability for more efficient routing

¹ Aurelia is a genus of jellyfishes, whose shape is similar to our proposed overlay structure. Therefore, we name it by Aurelia.

and robust indexing. In Aurelia, peers maintain different sizes of routing table and index range according to their available capacity. Powerful nodes can achieve faster routing and quicker searching. The routing table maintenance is based on the multicast tree embedded in the overlay. Aurelia also preserves the data locality by abandoning the using of hash functions for data distribution and each node is responsible for a continuous range of values. This property makes it easy to support complex queries. The performance of Aurelia is evaluated through simulation. Experimental results validate the scalability and efficiency of Aurelia.

The remainder of the paper is organized as follows. We discuss the related work in Section 2 and then describe the detailed design of Aurelia in Section 3. In the following section 4, experimental results from the simulation are presented and analyzed. Finally, we conclude in Section 5.

2 Related Work

In the design of overlay protocols, there are only a few designs addressing both data locality and node heterogeneity. Existing DHT designs (Chord[2], Pastry[3], etc) use randomized hashing for data placement, but hashing will destroy the data locality. To preserve data locality in DHT, order-preserving hash function ([4], [5], etc) or another indexing layer (e.g., SFC[6], PHT [7]) was introduced. However, the design is likely to be complicated and to result in bad performance. SkipNet [8] is another kind of structured overlay, which provides data locality by organizing data by string names, but it suffers from the problem of load balancing. Additionally, they all ignore node heterogeneity in their design.

Previous proposals to utilize node heterogeneity are mainly based on the concept of “supernode”. Nodes are simply classified into two categories: supernode or not. The system availability and performance greatly depend on the supernodes, and is contrary to the P2P design principle. In [9], Hu et al propose a new capacity-aware DHT design, in which the nodes can be further divided into different levels. However, due to node dynamism, it cannot perform well under non-uniform node distribution.

3 System Design

Like Chord, Aurelia organizes nodes into a circular ring that corresponds to the ID space $[0, 2^{128} - 1]$. Each node is assigned a unique identifier. However, we abandon the using of hashing to distribute the data objects across the nodes for it will destroy the data locality. Instead, each node of Aurelia is responsible for a contiguous range of data value, and Aurelia simply maps the data object to the node according to its value. In this way, the data locality can be preserved and range queries for a continuous data range become efficient.

To exploit the node heterogeneity, Aurelia differentiates the nodes according to their capacity. A node n 's capacity C_n is determined by its bandwidth B ,

storage S , CPU processing capability P and the node lifetime L . We can represent it as $C_n = f(B, S, P) \times g(L)$, where f, g are monotonic increasing functions. For simplicity, we adopt a similar approach as [9]. Nodes are classified into levels (e.g., 0, 1, ... n, among them, 0 is the top level). The size of routing table and the index range of an Aurelia node are proportional to the node's capacity level. On the extreme, one very powerful node can have routes to all the nodes and host the full data index, just like Napster.

In the design of Aurelia, the main challenge is to maintain such a big routing table efficiently in a large system. There are two basic approaches: one is by periodical probing of all the nodes in the routing table entries, which will cause huge traffic when the table size is big; another is by event notification, when the node changes its status, its neighbors or itself can broadcast the event to all the related nodes. The latter is more scalable than the former. Aurelia adopts the latter approach. To avoid too many redundant messages, Aurelia uses multicast to notify all the relevant nodes when nodes join, leave, fail or change status. However, it is not necessary for Aurelia to explicitly maintain the multicast tree, because it is already embedded in the overlay. In the following sections, we will introduce the design of Aurelia in details.

3.1 Identification Scheme

Initially, the node ID can be assigned by uniform hashing, through which the nodes are distributed evenly across the ring. The node ID is divided into three parts: range ID, random bits and level ID. For example, for the node 001010...1010, its range ID and level ID are 001 and 10 respectively, and the bits in between are random bits.

The range ID is the first r -bit prefix of the node ID, which defines the index range that the node will be responsible for. All the data keys whose first r -bit prefix is the same as the range ID will be placed in this node. The length of range ID determines the size of index range of that node if the objects are uniformly distributed. The nodes can choose a suitable length based on its capacity. When the node feels that the query workload is beyond its capacity, it can reduce the responsible index range by extending the length of range ID. Formally, supposing the range of the ring is mapped to $[0, 1]$, the node with range ID of $b_0b_1 \dots b_{r-1}$ will take charge of the range:

$$Range(b_0b_1 \dots b_{r-1}) = \left[\frac{\sum_{i=0}^{r-1} b_i \times 2^{r-i-1}}{2^r}, \frac{\sum_{i=0}^{r-1} b_i \times 2^{r-i-1} + 1}{2^r} \right)$$

The level ID is the l -bit suffix of the node ID, which represents the node capacity level. The level ID and routing range together determine the routing table size of the node. For a node n , only the nodes whose node ID is within the routing range and the last l bits are the same as its level ID will appear in its routing table. Supposing that nodes are uniformly distributed in the ID space, if there are N nodes in the system, the size of routing table is $N/2^l$. The shorter the level ID, the bigger the routing table. In the extreme case, if the length of level ID is zero, the routing table will include all the nodes in the system.

3.2 Routing Strategy

In case of uniform node distribution in the ring, the routing scheme of Aurelia will be similar to SmartBoa [9]. However, with node dynamism or load balancing reasons (e.g., under-loaded nodes quit their current position and rejoin the heavily-loaded region), the node distribution in the ring will become non-uniform. In such scenarios, SmartBoa’s routing scheme will become inefficient. For the region with dense node distribution, more hops will be required than expected (as illustrated in Fig.1(a)). This drawback leads to the design of Aurelia.

The basic idea of Aurelia is to adjust the span of routing pointers adapting to the node distribution. In the sparse region, there are fewer routing pointers; while in the dense region, more routing pointers will be allocated for fast routing. The density of routing pointers is controlled by adjustment of level ID. The node seems like hosting multiple virtual nodes with different level ID and routing range. However, these virtual nodes share the same node ID. Fig.1(b) gives a simple example of Aurelia’s routing pointers.

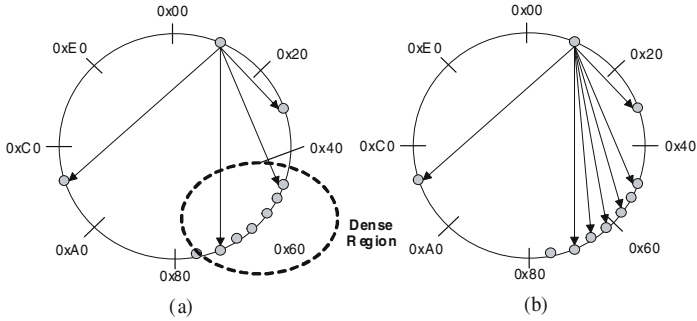


Fig. 1. Routing under non-uniform node distribution (a)SmartBoa’s routing pointers (b)Aurelia’s routing pointers

As to the routing strategy, Aurelia performs in a greedy-like style. Given a target key, the node always selects the nearest one in the routing table to forward it. For the powerful nodes, the big routing table enables them to complete the routing even in 1 hop.

To build such kind of routing table, a big challenge is to maintain the node-count distribution in a decentralized way. Aurelia adopts the technique of sampling to collect the statistical information and build the approximate node-count histogram locally. Each node in Aurelia maintains a leaf set with the size of k , which records the node’s k nearest neighbors in the ID space. Normally, there will be $k/2$ nodes on each side of the node. Periodically, the node will probe the existence of the k neighbors and adjust the entries in case of node joining or leaving. If denoting the distance between the node’s rightmost neighbor and the leftmost neighbor in the ring as d , then the node i can produce a local estimate D_i about node density in its nearby region, which is represented as $D_i = \frac{k+1}{d}$.

In addition to producing local estimate, the node also periodically makes sampling uniformly in the ring. According to the collected information, the node will choose the most recent statistical data to produce the node-count histogram. After the histogram is built and normalized, the node can adjust its routing pointers in the following way:

Each block in the histogram represents a continuous region with the size of $1/2^k$ of the whole ID space. Based on the ratio between the block's density level and the average density level, the length of virtual nodes' level ID will be increased or decreased accordingly. Such adjustment will impact the density of routing pointers in different routing ranges. For sparse routing range, fewer routing pointers are needed, so the level ID of virtual node in that region can be extended. On the contrary, for dense routing range, more routing pointers will be included by shortening the level ID. However, the extension and reduction of the level ID cannot deviate too much from its real level. Otherwise, it may cause problems in some situations. For example, if a weak node hosts a virtual node with high level and tiny routing range, it will join the high-level node group, which will result great update traffic beyond its capacity.

3.3 Routing Table Maintenance

The routing table of Aurelia includes four parts: routing pointers, top node pointers, leafset pointers and skip pointers.

The "routing pointers" part contains the nodes whose last l -bit suffix equals to the level ID and the node ID is within the routing range. The number of routing pointers scales nearly linearly with the increasing of node capacity. The "top node pointers" part is composed of the nodes whose level ID is the shortest suffix of current node's level ID. Top nodes are often powerful nodes that hold more routing pointers. The "leafset pointers" part includes the neighbors of current nodes in the ring. For the weak node's routing step span is still too large, there may be many nodes between two adjacent routing pointers. For fast routing, "skip pointers" are maintained in the region between the node's first right routing pointer and left routing pointer. Each of the skip pointers will skip 2^i ($i = 0, 1, \dots$) nodes.

To maintain the routing pointers in a scalable fashion, Aurelia adopts multicast to distribute the events of node join, leave or status change. The multicast tree doesn't require explicit management. It is based on the level ID and routing range to disseminate the event from high-level nodes to low-level nodes.

The details of the multicast process are as follows: when a node joins or changes its status, it will first forward the event with the node ID to one of its top nodes randomly. Then this top node will multicast the event to all the nodes whose level ID is the suffix of the reported node ID. But for the virtual node, if an announced node ID is not within the routing range it wants to be notified, the event will be ignored. In every step of the multicast process, the node that receives the event will first send it to the next lower level, then notifies the other nodes with the same level ID. By this, we can guarantee the event to be exactly delivered from the high-level nodes to the low-level nodes. When a node

leaves, its predecessor will detect this event and help to notify the top nodes and propagates the change information to all the related nodes.

A node can adaptively adjust its level and routing table according to the workload it experiences. In the high-level position, the node can establish a big routing table and do quick routing, but it is at the cost of more processing and bandwidth consumption.

3.4 Data Registration

To preserve the data locality, Aurelia avoids using hashing to distribute the data objects. Each node of Aurelia will be responsible for a contiguous range region, which is determined by its range ID. The data registration is simply based on the data value. If the value falls into the range of one node, then the node will hold the data object or the pointer to it. In this way, the objects with adjacent value will be assigned to the same node or the nearby nodes. This property makes range query to be efficiently supported in Aurelia. When publishing a new data object, its value is first normalized into the range $[0,1]$. Given that the normalized value is v , we can approximately represent v by a binary string $b_0b_1\dots b_{n-1}$ (n is the maximum ID length). They satisfy the following relation: $v \approx \frac{b_0}{2} + \frac{b_1}{2^2} + \dots + \frac{b_{n-1}}{2^{n-1}}$.

The binary string is the key of this data object, then this data will be registered into all the nodes whose range ID is a prefix of the key. In this way, one data object will have multiple index entries. It provides a kind of replication for index information and makes the indexing service more robust. Even when some nodes hosting the index information leave the system, the lookup may still be successful. For example, in Fig.2, two nodes are responsible for different range sizes in the ring. The index range of node 010xxxxx is covered by node 01xxxxxx.

The detailed registration process is as follows: Similar to the “top node pointers” in the routing table, every node also maintains “top index pointers”, whose range ID is the prefix of the node’s range ID and index range is not covered by any other nodes. This means that it keeps a bigger range that covers the range of this node. During data publishing, the registration message is firstly routed to the node whose ID is nearest to the data key; then the node will select a top

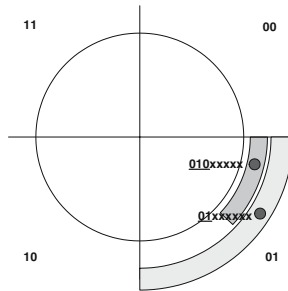


Fig. 2. Index Range Coverage

index node randomly and forward the message to it. This top index node will be responsible for multicasting the message to all the nodes whose range is covered by it. The multicast algorithm is similar to the above-mentioned multicast algorithm for routing table maintenance, except that it is based on the range ID for multicasting. When a node feels overloaded by query traffic, it can adjust its range ID to change index range.

4 Experiments

We evaluate the Aurelia protocol by simulation on a Sun Enterprise E4500 server (with 12 UltraSPARC-II 400MHz CPUs, 8GB RAM and 1Gbps network bandwidth). To understand Aurelia's routing performance in practice, we simulate a network with $N = 10,000$ nodes and use the routing hops during a lookup operation as the metric. We assume node distribution in the ring follows a Zipf-like distribution. Each node in an experiment selects a random set of keys to query from the system, and we measure the hops required to resolve each query. To investigate the effects of node heterogeneity, the node capacity distribution is based on the measurement results of Gnutella in [1]. (as shown in Fig.3 (a))

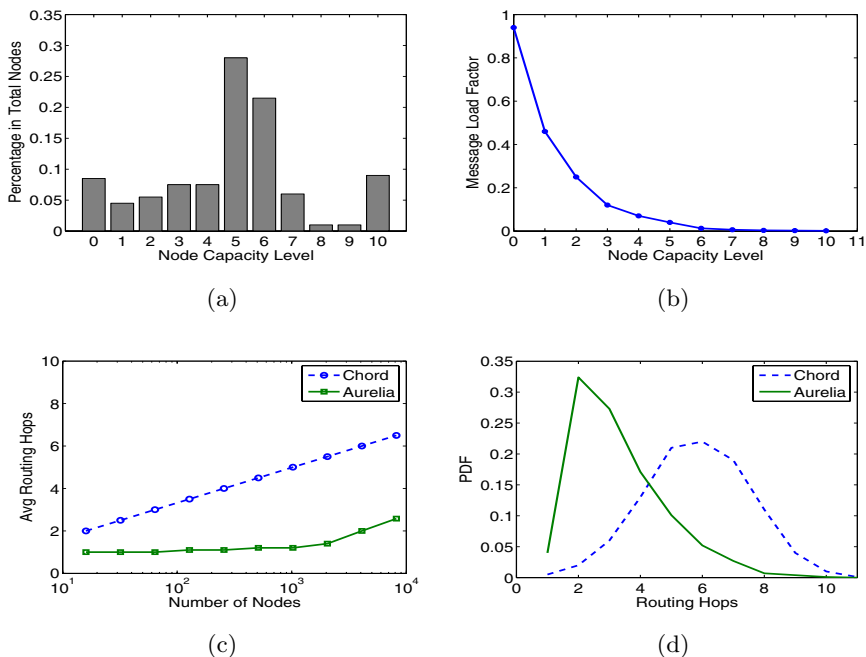


Fig. 3. (a)Node capacity distribution; (b)Message load factor of nodes with different capacity levels (c)Average routing hops as a function of network size (d) PDF(Probability Distribution Function) of the routing hops for a network with 10000 nodes

Fig.3 (b) describes the message load factor of nodes with different capacity levels. High-level nodes are likely to have more message traffic. In our experiments, we take Chord[2] as a benchmark for routing performance comparison. Fig.3 (c) plots average routing hops as a function of network size. When the network size is small, even the weak nodes can maintain the routes to all nodes, and the average routing hops is near 1. With the increasing of node number, even the top nodes cannot maintain the full routing table, and the average routing hops will increase accordingly. However, the routing efficiency of Aurelia is still better than Chord. Fig.3 (d) plots the PDF(Probability Distribution Function) of the routing hops for a network with 10000 nodes. It is observed that most lookups can be completed within 5 hops.

5 Conclusion

We have presented a locality-preserving heterogeneous overlay design called Aurelia in this paper. It fully utilizes node heterogeneity to achieve fast routing and robust indexing. By guaranteeing data locality, complex queries like range query can be supported more efficiently. In the future, we will try to analyze its performance from theoretical perspective and conduct experimental implementation on the PlanetLab[10].

References

1. Saroiu, S.: Measurement and analysis of internet content delivery systems. Doctoral Dissertation, University of Washington (2004)
2. Stoica, I., et al.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. ACM (SIGCOMM'01), San Diego, CA (2001) 149–160
3. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proc. IFIPACM International Conference on Distributed Systems Platforms (Middleware'01), Heidelberg, Germany (2001) 329–350
4. Gupta, A., Agrawal, D., Abbadi, A.E.: Approximate range selection queries in peer-to-peer systems. In: Proc. the First Biennial Conference on Innovative Data System Research (CIDR'03), Asilomar, CA, USA (2003)
5. Cai, M., Frank, M., Chen, J., Szekely, P.: Maan: A multi-attribute addressable network for grid information services. In: Proc. 4th International Workshop on Grid Computing (Grid'03), Phoenix, Arizona (2003)
6. Schmidt, C., Parashar, M.: Enabling flexible queries with guarantees in p2p systems. Internet Computing Journal, Vol.8, No.3 (2004)
7. Ramabhadran, S., Ratnasamy, S., Hellerstein, J.M., Shenker, S.: Brief announcement: Prefix hash tree. In: Proc. ACM (PODC'04), St. Johns, Canada (2004)
8. Harvey, N.J.A., et al.: Skipnet: A scalable overlay network with practical locality properties. In: Proc. Fourth USENIX Symposium on Internet Technologies and Systems (USITS'03), Seattle, WA (2003)
9. Hu, J., et al.: Smartboa: Constructing p2p overlay network in the heterogeneous internet using irregular routing tables. In: Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04), San Diego, CA, USA (2004)
10. Planetlab: <http://www.planet-lab.org/>