Arpeggio

Efficient Metadata-based Searching and File Transfer with DHTs

Arpeggio is a peer-to-peer file-sharing network based on the Chord distributed hash table. Queries for files whose metadata matches a certain criterion are performed efficiently by using a **distributed keyword-set index**, augmented with **index-side filtering**. We introduce **metadata gateways**, a technique for minimizing index maintenance overhead. *Arpeggio* also uses the DHT for **indirect storage** of file contents, maintaining pointers from content to the live peers that provide it. Finally, we introduce **postfetching**, a technique that uses information in the index to improve the availability of rare files. The result is a system that provides efficient query operations with the scalability and reliability advantages of full decentralization, and a content distribution system tuned to the requirements of a peer-to-peer file-sharing network.

Idea

DHTs offer efficient *lookup*, but not efficient *search*

DHTs can enhance content *availability* and distribution *performance*

Indexing

- Insertion
  - Copies of the metadata block containing the full keyword set of a content object and information on how to retrieve it are inserted under every subset of the set of keywords of size up to some *k*.

- Searching
  - The largest possible subset of the query is used to look up an index node. The index node is passed the full query, which it uses to filter its index and return exactly the relevant results.

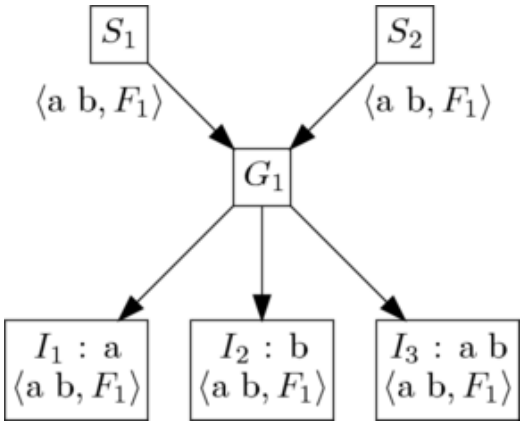
```
INSERT(keyword-set S, data D)
1  for each subset s ⊆ S with |s| ≤ k
2        do m ← ⟨S, HASH(D)⟩
3           ▷ m is the data's metadata block
4           I ← LOOKUP(HASH(s))
5           ▷ I is the index node for HASH(s)
6           I.PUT(m)
```

```
SEARCH (query Q)
1  ▷ Executed on requesting node
2  S ← KEYWORDS(Q)
3  for some s ⊆ S          (|s| ≤ k)
4  I ← LOOKUP(HASH(s))
5  ▷ I is the index node for s
6  return I.FILTERED-GET(Q)
```

```
FILTERED-GET (query Q)
1  ▷ Executed on index node
2  R ← ∅
3  for all metadata blocks m in the local index
4        do if m matches Q
5              then R ← R ∪ m
6  return R
```

Maintenance

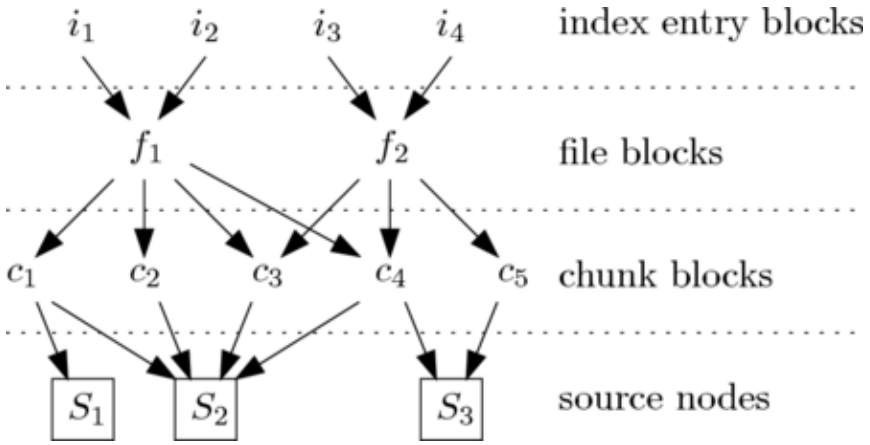
- Expiration
  - Metadata is expired to handle shifting network contents. However, expiration time is long so that postfetching can be effective.
- Metadata Gateways
  - Metadata blocks are inserted in to the network via metadata gateways instead of directly in order to aggregate the high insertion cost. This way, the inserting node only has to send the new metadata block to the gateway, which will in turn register it with the index nodes, but only if necessary.



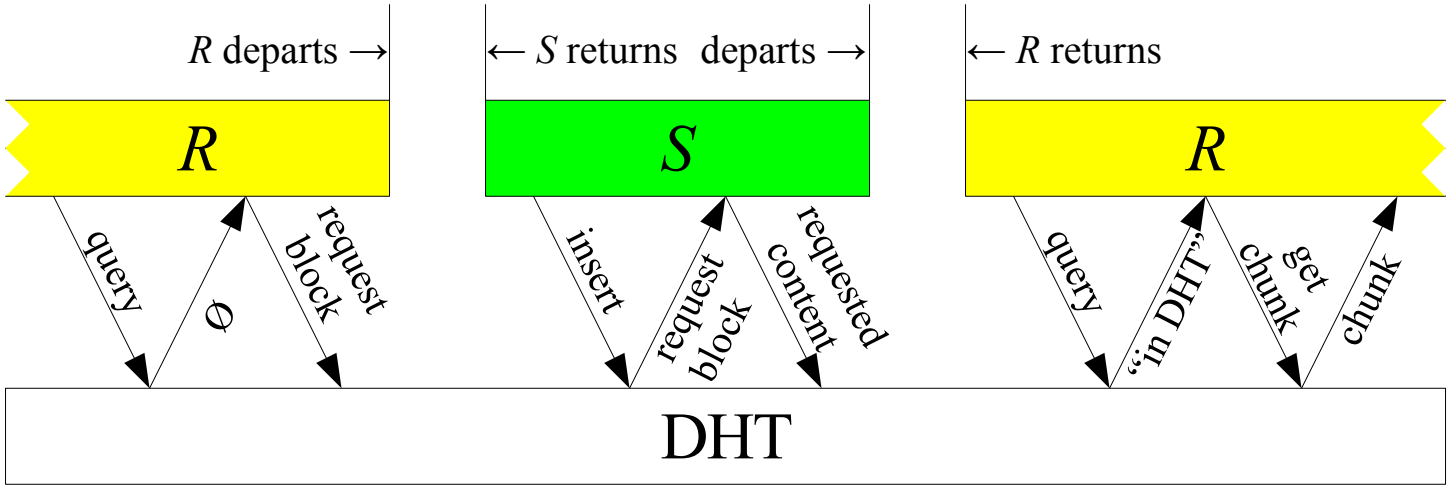
- Replication
  - Index data is lazily replicated because only weak replica consistency is necessary. Erasure coding is not used because of the high ratio of queries to insertions.
- Incremental Responsibility
  - Nodes do not immediately join the DHT, but proxy through another node until they have proven stable enough to join the ring. As their stability increases, they can join the ring multiple times to shoulder more of the load.

Content Distribution

- Segmentation
  - Files are divided into dynamically-sized chunks, which can themselves be shared among similar files. Peers share chunks and files are represented as lists of chunk block hashes.



- Postfetching
  - Rare files are cached on other peers when demand is indicated by request blocks.



Conclusion

Arpeggio differs from previous peer-to-peer file sharing systems in that it implements both a metadata indexing system and a content distribution system using distributed hash tables. We extend the standard DHT interface to support not only lookup by key but **complex search queries**. **Keyword-set indexing** and extensive network-side processing in the form of **index-side filtering**, **metadata gateways**, and **expiration** are used to ameliorate the scalability problems inherent in distributed document indexing. We introduce a content-distribution system based on **indirect DHT storage** that uses **chunking** to leverage file similarity, and thereby optimize availability and transfer speed. Availability is further enhanced with **postfetching**, which uses cache space on other peers to replicate rare but demanded files.