Midterm Examination

CSE 455 / CIS 555 – Internet and Web Systems

Spring 2008

Zachary Ives

Name: Answer Key

6 questions, 100 pts, 80 minutes

1. (20 pts) Explain the basic (a) lookup and (b) file/object replication capabilities in the Chord DHT versus the "unstructured" Gnutella peer-to-peer network.

Chord uses a "ring" with values from 0 to 2^{160}-1 as its basic abstraction. Each node's IP address is hashed and placed in the ring; values are also hashed by their keys, and "owned by" the closest node whose hash value is greater than or equal to the key's hash. Nodes in Chord have lg(n) entries in a "finger" table, pointing to a node halfway around the ring, $1/4^{th}$ of the way, $1/8^{th}$ of the way, …, all the way to the successor. Data is also **replicated** on the lg(n) successors of the owner.

Lookup for a key *k* proceeds by doing a search through the finger table, using the destination nodes' keys as a "pivot" in a fashion similar to a search of a tree. The goal is to find the *predecessor* of the owner of *k*. If this predecessor is not reachable in one hop, the search request will be forwarded to the nearest node to the predecessor, and the process will repeat recursively.

Gnutella relies on flooding (usually to some time-to-live *n*) to do **lookups**. Once an object is looked up, it is typically **replicated** on the requesting node, which should make it available for future lookups.

(c) What are the guarantees provided by each P2P network?

Chord guarantees that, in the absence of **churn** (nodes joining and leaving) and dropped messages, if a key exists within the DHT, its value will be returned. Gnutella guarantees that, in the absence of churn and dropped messages, an object will be retrieved if it is within the time-to-live range.

(d) What are the reasons one would prefer an unstructured network vs. a structured one?

Unstructured networks like Gnutella are less sensitive to churn; structured networks like Chord must continuously ship state back and forth as nodes join and leave.

2. *(20 pts)* Suppose we were to replace the DNS lookup system with one based on the forward axes of XPath (using the same hierarchical structure and naming). For instance, www.cis.upenn.edu/~zives/cis555 might become edu/upenn/cis/www/~zives/cis555. Assume we have /, //, and * but no [] predicates. (a) How would we need to change the lookup system in order to make this work correctly?

This would require substantially more work on the part of the lookup system: instead of matching a single path, we instead need to match a path against a tree, returning a node set. Either the node doing the lookup would have to recursively traverse the hierarchy of nodes on step at a time, at each level collecting all node sets that match, or it would need to offload this work to the DNS nodes (who in turn would need to perform expensive recursive traversals). Internally, this could be done using a scheme similar to XFilter.

Caching of names would be significantly more complex, as there might be XPath expressions that partly match but aren't exactly the same. Segmenting the XPaths, which consist of a portion that matches on the domain name hierarchy and a portion that matches on paths in the filesystem of a machine, becomes tricky.

(b) Would this represent an improvement over the existing DNS naming scheme, or a step backwards? Why?

It gives a great deal more flexibility, but it changes the semantics of lookup significantly (a set of nodes, potentially heterogeneous, is quite different from a single node), and it incurs greater lookup cost. It is also more vulnerable to overload or denial-of-service attacks.

3. *(15 pts)* Given two XML files with the following form:

Students.xml

```
<students>
  <student>
    <name>{person}</name>
    <sid>{student ID}</sid>
  </student>
…
</students>
```

SSNs.xml

```
<ssns>
```

```
  <mapping>
    <sid> { student ID } </sid>
    <ssn> { SSN } </ssn>
  </mapping>
  …
</ssns>
```

Write an XQuery that outputs each student, the student's name, and the student's SSN.

```
for     $st in doc("Students.xml")/students/student,
        $sn in $st/name,
        $sid in $st/sid
return <student>
        { $sn }
        { doc("SSNs.xml")/ssns/mapping[sid=$sid]/ssn }
        </student>
```

OR

```
for     $st in doc("Students.xml")/students/student,
        $sn in $st/name,
        $sid in $st/sid,
        $sm in doc("SSNs.xml")/ssns/mapping,
        $sid2 in $sm/sid,
        $ssn in $sm/ssn
where $sid = $sid2
return <student>
        { $sn }
        { $ssn }
        </student>
```

(and many other variants)

4. (15 pts) Explain the roles of (a) WSDL and (b) SOAP in implementing remote procedure calls. What are the corresponding concepts in Java RMI?

(a) WSDL is the interface definition language (IDL) for Web services. It specifies what functions are available, what their RPC bindings are, what the parameters, datatypes, and return values are, and so forth. It is analogous to the Java interface definitions that are used for RMI.

(b) SOAP is the data/error serialization layer. Within the SOAP envelope we get constraints on what tags must be understood/implemented, sets of parameters, and so on. The return value is also typically in SOAP, and it may include error information. SOAP is analogous to Java serialization.

5. (10 pts) We have discussed the internal architecture of Google to a significant extent in this course: it has many replica sets, each of which may be independently refreshed during a new crawl of the Web. Such an architecture wasn't well-suited for an EBAY or an Amazon, but it worked very well for search. Discuss how well it fits for a system like YouTube.

Many of the basic features are quite well-suited for YouTube, where videos are essentially static for all time. One can partition or "stripe" videos by ID (and according to load distribution) across different machines, and can assume that these machines will not need to communicate with one another. Each machine will independently serve videos for its partition; it will have many replicas. (Replication could be done, e.g., via Google's replicating distributed filesystem.) YouTube does not really need an offline crawler, as videos are directly uploaded.

There is one aspect that is significantly different: YouTube places much more I/O demands on machines and the network. Most likely, more replicas of each node must be created, and the machines and network would need to have higher throughput. Many data centers would be geographically distributed, and the goal would be to forward requests to a "nearby" node.

6. (20 pts) Assume we have a cluster of machines, each containing lists of URLs crawled. Describe how one would use the MapReduce programming model to count how many documents named "index.html" have been crawled. Include pseudocode for the map and reduce components.

```
map(key,log):       iterate through log entries
                    if entry ends with "index.html" { emit (key="index.html" , val=1) }

reduce(key,vals):   foreach count in vals { sum += count }
                    emit sum
```