Midterm Examination

CIS 455 / 555 – Internet and Web Systems

Spring 2010

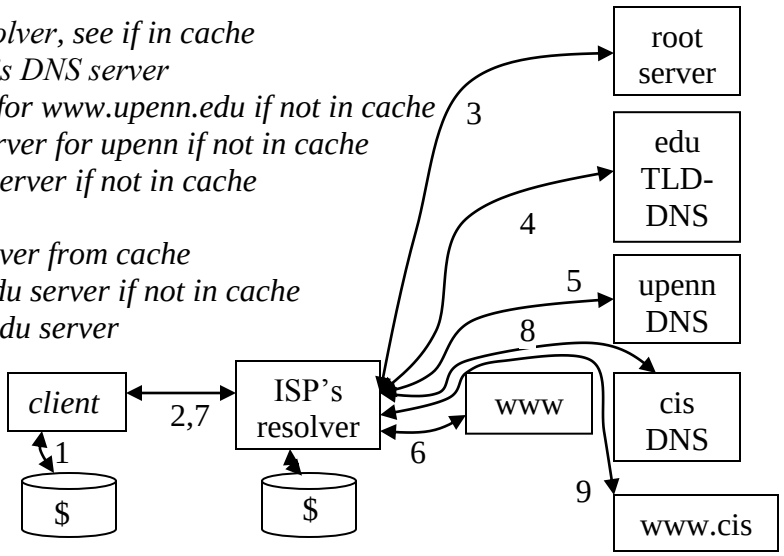
Zachary Ives

Name: Sample Answer Key

7 questions, 90 pts, 80 minutes

1. *(15 pts)* Describe, showing with a combination of pictures and steps, the process for a client machine (hooked up to an ISP) looking up the domain names **www.upenn.edu** followed by **www.cis.upenn.edu**. Consider where caching can be used.

1. Consult client's DNS resolver, see if in cache
2. Recursive request to ISP's DNS server
3. ISP: Consult root server for www.upenn.edu if not in cache
4. ISP: Consult edu TLD server for upenn if not in cache
5. ISP: Consult upenn.edu server if not in cache
6. Contact www.upenn.edu
7. ISP: Fetch upenn.edu server from cache
8. ISP: Consult cis.upenn.edu server if not in cache
9. Contact www.cis.upenn.edu server



2. (15 pts) Given XML files of the form:

source.xml

```
<rss>
  <title> {channel title} </title>
  <link> {web url} </link>
  <description> { brief description } </description>
  <item>
    <title>{article 1 title}</title>
    <link>{URL 1}</link>
    <author>{person 1}</author>
    <description> { article 1 description } </description>
  </item>
  <item>
    <title>{article 2 title}</title>
    <link>{URL 2}</link>
    <author>{person 2}</author>
    <description> { article 2 description } </description>
  </item>
…
</rss>
```

Write an XSLT stylesheet that outputs an HTML document of the form:

```
<html>
<head>
<title> {channel title} </title>
</head>
<body>
<h1> {channel title} </h1>

<h2> {article 1 title} </h2>
<p> {article 1 description} </p>

<h2> {article 2 title} </h2>
<p> {article 2 description} </p>

…
</body>
</html>
```

One of several options:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/rss">
 <html>
 <head>
 <title><xsl:apply-templates/></title>
 </head>
 <body>
  <h1><xsl:apply-templates/></h1>
  <xsl:for-each select="item">
   <h2><xsl:apply-templates/></h2>
   <p><xsl:value-of select="description/text()"/></p>
  </xsl:for-each>
 </body>
 </html>
</xsl:template>

<xsl:template match="title">
 <xsl:value-of select="text()"/>
</xsl:template>

</xsl:stylesheet>
```

3. (10 pts) List **at least two cases** where the Google File System offers *reduced capabilities or guarantees* versus a standard file system (e.g., NFS, your favorite local file system). For each such instance, explain *how this helps the system provide better reliability or scalability*.

1. *GFS offers reduced guarantees about write semantics – a record will be written "at least once" in the presence of failure. This enables GFS to more efficiently handle failures: if a subset of the machines fail to write, GFS can simply retry. Some machines will have 2 copies of the write, whereas others only have one copy.*

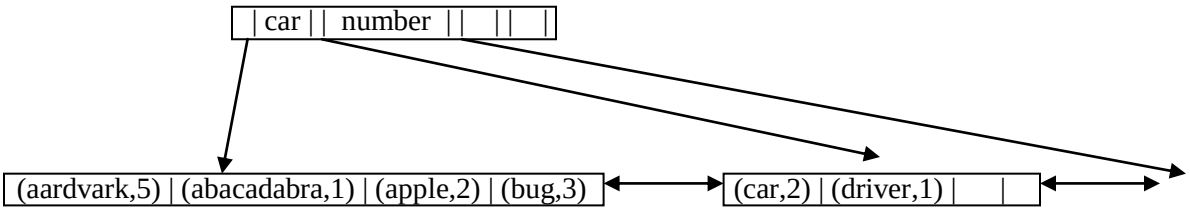
2. *GFS record appends are nondeterministic with respect to position. If multiple appends are requested concurrently, the primary chunkserver will interleave them in a consistent but unpredictable way. This reduces the amount of coordination during concurrent writes, resulting in better performance.*

3. *GFS does not provide guarantees about data integrity or consistency. Applications are expected to have checksums and optional checkpoints, as mechanisms for enabling consistency. This makes failure handling less expensive at the GFS level.*

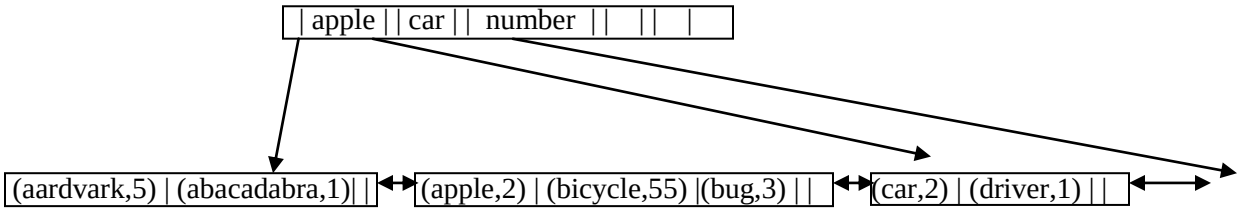
4. (10 pts) Explain the key differences between the REST and SOAP Web Service calling and parameter passing conventions. You do not need to discuss return values and codes.

REST, while not truly a standard, uses URLs to specify a function invocation (a function is simply given a pathname), and POST or GET parameters to pass in parameters. SOAP uses a URI plus a "SOAP envelope" (an XML format) to invoke the function and pass in parameters. A SOAP Web Service typically needs an accompanying XML Schema and WSDL interface definition.

5. (10 pts) The B+ Tree is a balanced tree index structure. Assume you are given the following B+ Tree inverted index, with keywords and document IDs:



Redraw the B+ Tree after the keyword "bicycle" with document ID 55 is inserted.



6. *(15 pts)* Assume we are given a task to find the most popular Web page *at each Web site* (domain name), given a log of access requests, each of the form "{domain} {page} {requestor IP}". Show pseudocode for **map** and **reduce** functions for a MapReduce operation to do this computation.

SINGLE-PASS VERSION

```
map(domain, record) {
        emit(record.domain, record.page);
}

reduce(domain, set< page > occurrences) {
        hashmap h = new hashmap<page, count>;
        foreach (o in occurrences) {
                if exists h[o] then h[o]++ else h[o] = 1;
        }
        int count = -1;
        string page = "";
        foreach (k in h.keyset) {
                if (h[k] > count) {
                        page = k;
                        count = h[k];
                }
        }
        emit(domain, page);
}
```

TWO-PASS VERSION

First pass:

```
map(domain, record) {
        emit(new pair(record.domain, record.page), 1);
}

reduce(pair<domain,page> url, set<integer> occurrences) {
        count = occurrences.size;
        emit(url.domain, new pair(url.page,count));
}
```

Second pass:

```
map(domain, pair<page,count> url) {
        emit(domain, url);
}

reduce(domain, set<pair<page,count>> urls) {
        int count = -1;
        string page = “”;
        foreach (u : urls) {
                if (u.count > count) {
                        page = u.page;
                        count = u.count;
                }
        }
        emit(domain, page);
}
```

7. *(15 pts)* Assume we are given a task to find the most popular Web page *at each Web site* (domain name), given an in-memory ArrayList of Log elements, where Log is a class with three member variables { String domain; String page; String ip; }. Show how one would implement this task using **Pastry**, by providing pseudocode for the **Message** class (or subclasses) and for the **deliver** method.

There are many possible implementations. A challenge is knowing when "end of input" is reached, such that it is time to produce an aggregate value (sum or max). We didn't actually consider this in grading, but focused on your dataflow. To give a sense of the full problem, we show here a full two-stage implementation that assumes data is arbitrarily partitioned across all nodes at the start, and which uses Pastry's broadcast facilities. Every receiver waits until it has heard an end-of-stream message from all nodes that sent it data. A more concise one-stage implementation can also be defined, much like the one for MapReduce above, where we only have one level of hashing (by domain) and everything else is in memory.

```
class Message {
        enum type {SCAN_LOG, PAGE_HIT, COUNTS, PAGE_COUNT, MAX };
        Object key, payload;
        NodeHandle sender;
}

void deliver(Message m) {
        myHandle = getNodeHandle();
        switch (m.type) {
        case SCAN_LOG:                  // Send each occurrence
                for (Log l : logEntries)
                        route(l.domain + l.page, new Message(PAGE_HIT, new
                                Pair(l.domain, l.page), l, myHandle));
                broadcast(new Message(COUNTS, null, null, myHandle)); // Sender done
        case PAGE_HIT:                  // Receive + incr. accumulate page hits
                logSenders.add(m.sender);
                if exists pageCounts[m.key] pageCounts[m.key]++ else
                        pageCounts[m.key] = 1
        case COUNTS:                    // Wait for all senders, then send counts
                doneScanning.add(m.sender);
                if (logSenders == doneScanning)
                        for (Page p : pageCounts.keys) {
                                domain = p.first;
                                route (domain, new Message(PAGE_COUNT, domain, new
                                        Pair(p.second, pageCounts[p]), myHandle);
                        }
```

```
            broadcast(new Message(MAX, null, null, myHandle));        // sender done
        case PAGE_COUNT:
            countSenders.add(m.sender);
            domainCounts[m.key].add(m.payload);
        case MAX:
            doneCounting.add(m.sender);
            if (countSenders == doneCounting)
                for (d : domainCounts.keys) {
                    int count = -1;
                    string page = "";
                    for (Pair <u, c> : domainCounts[d]) {
                        if (c > count) {
                            page = u;
                            count = c;
                        }
                    }
                    output c;
                }
}
```