Distributed Suffix Array Construction

Huibin Shen

Department of Computer Science
University of Helsinki

String Processing Project
April 25, 2012
Outline

1. Problem
2. Implementation
3. Experiments
4. Conclusion
Suffix array definition

The suffix array of a text $T$ is a lexicographically ordered array of the set $T_{[0...n]}$ of all suffixes of $T$. More precisely, the suffix array is an array $SA[0...n]$ of integers containing a permutation of set $[0...n]$ such that

$$T_{SA[0]} < T_{SA[1]} < \cdots < T_{SA[n]}.$$

Example: The suffix array of the text $T = banana$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$SA[i]$</th>
<th>$T_{SA[i]}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>$$$</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>a$</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>ana$</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>anana$</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>banana$</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>na$</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>nana$</td>
</tr>
</tbody>
</table>
Suffix array construction algorithm

Existing suffix array construction algorithm:

- Prefix Doubling, $O(n \log n)$.
- DC3, $O(n)$.
- SAIS, $O(n)$.

All on one computer!
Deploy the suffix array construction on clusters!

- Each node of the cluster becomes a bucket for a subset of all the suffixes.
- Each node sorts the subset of suffixes independently.
- Merge the result of each node.

Diagram is shown on next slide.
suffixes of T: \{banana$, anana$, nana$, ana$, na$, a$, $\}

sorted suffixes of T: \{$, a$, ana$, anana$, banana$, na$, nana$\}
Besides the distributed suffix array construction algorithm, a linear time suffix array construction algorithm DC3 is also implemented for comparison.

All codes are written in python.

Important pieces of code of distributed suffix array construction are shown here, while codes for DC3 are ignored.
# deploy one task on one node

```python
def deploy(node, left, right):
    os.system("ssh %s 'cd /home/fs/hzshen/string_proj/; python on_one_node.py %s %s %s'" % (node, node, left, right))
```

# deploy tasks on nodes

```python
for i in range(n_nodes):
    node, load = nodes[i]
    # pid is the index of the left pivot for one bucket
    t.append(Process(target=deploy, args=(node, p_id, p_id+1)))
    p_id += 1
for p in t:
    p.start()
for thread in t:
    p.join()
```
# T is the original text, pivots is the list of pivots
# p1, p2 are the indices of left, right pivots
# pivots[p1] is the left pivot, pivots[p2] is the right pivot
# iteration indicates the recursion level
# tuples are like [('b','a',1),('a','n',2),...]
# output is the suffix array

def sortSuffixes(T, pivots, p1, p2, iteration, tuples, output):
    tuples = sorted(tuples, key = lambda x: str(x[0]) + str(x[1]))
    # partition filters out the tuples that are not belong to this bucket.
    # Divide tuples by the first two chars into small buckets until one
    buckets = partition(tuples, p1, p2)
    iteration += 1
    for i in xrange(len(buckets)):
        if len(buckets[i]) == 1:
            output.append(buckets[i][0][2])
        else:
            tuples2 = []  # go one level deeper
            for j in xrange(len(buckets[i])):
                tuples2.append((i, TT[buckets[i][j][2] + iteration-1], buckets[i][j][2]))
            sortSuffixes(TT, pivots, p1, p2, iteration, tuples2, output)
Time comparison between DC3 and distributed sorting with 104 nodes
Time comparison in distributed sorting with different nodes

<table>
<thead>
<tr>
<th>Input Size</th>
<th>Time in s</th>
</tr>
</thead>
<tbody>
<tr>
<td>5KB</td>
<td>80</td>
</tr>
<tr>
<td>10KB</td>
<td>82</td>
</tr>
<tr>
<td>25KB</td>
<td>84</td>
</tr>
<tr>
<td>50KB</td>
<td>86</td>
</tr>
<tr>
<td>100KB</td>
<td>88</td>
</tr>
<tr>
<td>250KB</td>
<td>90</td>
</tr>
<tr>
<td>500KB</td>
<td>92</td>
</tr>
<tr>
<td>1MB</td>
<td></td>
</tr>
<tr>
<td>2.5MB</td>
<td></td>
</tr>
</tbody>
</table>

Distributed Sorting 26 nodes
Distributed Sorting 52 nodes
Distributed Sorting 78 nodes
Distributed Sorting 104 nodes
Distributed Sorting 130 nodes
Space comparison between DC3 and distributed sorting with 104 nodes
Space comparison in distributed sorting with different nodes
Running time distribution of distributed sorting over 104 nodes

- 5KB
- 10KB
- 25KB
- 50KB
- 100KB
- 250KB
- 500KB
- 1MB
- 2.5MB
- 5MB
Do **not** use python when implementing string processing algorithm!
Dynamically construct pivots based on string distribution so that every node (buckets) receive as equal amount of task as possible.
Questions?