CS639: Data Management for Data Science

Lecture 8: Reasoning about Scale & The MapReduce Abstraction

Theodoros Rekatsinas
Logistics/Announcements

- Submission template for PA2
- Bonus problem for PA2
- Other questions on PA2?
Today’s Lecture

1. Scalability and Algorithmic Complexity

2. Data-Parallel Algorithms

3. The MapReduce Abstraction
1. Scalability and Algorithmic Complexity
What does scalable mean?

• Operationally:
  • Works even if the data does not fit in main memory
    • Use all available resources (cores/memory) on a single node (aka scale up)
    • Can make use of 1000s of cheap computers (cloud) – elastic (aka scale out)

• Algorithmically:
  • If you have N data items you should not perform more than $N^m$ operations (polynomial complexity)
  • In many cases it should be $N \times \log(N)$ operations (streaming or too large data)
  • If you have N data items, you must do no more than $N^m/k$ operations for some large k (k = number of cores/threads)
A sketch of algorithmic complexity

• Example: Find matching string sequences

• Given a set of string sequences
• Find all sequences equal to “GATTACGA”
Example: Find matching string sequences

GATTACGA
Example: Find matching string sequences

Time = 0: TACCTGCC ? GATTACGA
Example: Find matching string sequences

Time = 0: TACCTGCC ? GATTACGA
No move cursor to next data entry
Example: Find matching string sequences

Time = 1:  EFTAAGCA ? GATTACGA
No move cursor to next data entry
Example: Find matching string sequences

Time = 2: XXXXXXX ? GATTACGA
No move cursor to next data entry
Example: Find matching string sequences

Time = n:  GATTACGA ? GATTACGA
Yes! Output matching sequence
Example: Find matching string sequences

If we have 40 records we need to perform 40 comparisons
Example: Find matching string sequences

For N records we perform N comparisons

The algorithmic complexity is order N: O(N)
What if we knew the sequences are sorted increasing lexicographically?
What if we knew the sequences are sorted

Increasing Lexicographic Order

Time = 0: Start at 50% mark CTGTACA < GATTACGA
What if we knew the sequences are **sorted**

**Increasing Lexicographic Order**

**Time = 1:** Start at 50% mark $\text{CTGTACA} < \text{GATTACGA}$

Skip to 75% mark (you know your sequence is in the second half)
What if we knew the sequences are \textit{sorted}

\begin{itemize}
\item \textbf{Time} = 2: We are at the 75\% mark \texttt{TTGTCCA} > \texttt{GATTACGA}
\item Skip to 62.5\% mark Match: \texttt{GATTACGA} = \texttt{GATTACGA}
\item We find our sequence in three steps. Now we can scan entries sequentially.
\end{itemize}
What if we knew the sequences are sorted

Increasing Lexicographic Order

How many comparisons?
For N records we did log(N) comparisons
The algorithm has complexity O(log(N)) — much better scalability
2. Data-Parallel Algorithms
New task: Trim string sequences

- Given a set of string sequences
- Trim the final $n$ characters of each sequence
- Generate a new dataset
New task: Trim string sequences (last 3 chars)

Time = 0: TACCTG GCC -> TACCTG
New task: Trim string sequences (last 3 chars)

Time = 1: GATTCTGC -> GATTC
New task: Trim string sequences (last 3 chars)

Time = 2: \texttt{CCCGAAT} $\rightarrow$ \texttt{CCCG}

Can we use a data structure to speed this operation?
New task: Trim string sequences (last 3 chars)

Time = 2:    CCCGAAT -> CCCG
Can we use a data structure to speed this operation?
No. We have to touch every record! The task is O(N).
New task: Trim string sequences (last 3 chars)
New task: Trim string sequences (last 3 chars)
New task: Trim string sequences (last 3 chars)

Time = 1: Process first element of each group
New task: Trim string sequences (last 3 chars)

Time = 2:  Process second element of each group
New task: Trim string sequences (last 3 chars)

Time = 3: Process third element of each group
Etc.. How much time does this take?
New task: Trim string sequences (last 3 chars)

We only need $O(N/k)$ operations where $k$ is the number of groups (workers)
1. You are given a set of “reads”. You have to process them and generate a “write”

2. You distribute the reads among k computers (workers)
2. You distribute the reads among \( k \) computers (workers)

3. Apply function \( f \) to each read (for every item in each chunk)

4. Obtain a big distributed set of outputs
Applications of parallel algorithms

- Convert TIFF images to PNG
- Run thousands of simulations for different model parameters
- Find the most common word in each document
- Compute the word frequency of every word in a single document
- Etc....
Applications of parallel algorithms

• Convert TIFF images to PNG
• Run thousands of simulations for different model parameters
• Find the most common word in each document
• Compute the word frequency of every word in a single document
• Etc....

• There is a common pattern in all these applications
Applications of parallel algorithms

• A function that maps a string to a trimmed string
• A function that maps a TIFF images to a PNG image
• A function that maps a set of parameters to simulation results
• A function that maps a document to its most common word
• A function that maps a document to a histogram of word frequencies
Applications of parallel algorithms

• What if we want to compute the word frequency across *all* documents?
3. The MapReduce Abstraction
Compute the word frequency across 5M documents

```
for each doc return (word, freq) pairs
```

```
<table>
<thead>
<tr>
<th>map</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
</tr>
<tr>
<td>map</td>
</tr>
<tr>
<td>map</td>
</tr>
<tr>
<td>map</td>
</tr>
<tr>
<td>map</td>
</tr>
</tbody>
</table>
```

```
Distribute among k workers
```

```
Millions of Documents
```
Compute the word frequency **across** 5M documents

for each doc return (word, freq) pairs.

Then what?
Challenge: in this task

• How can we make sure that a single computer has access to every occurrence of a given word regardless of which document it appeared in?

• Ideas?
Compute the word frequency **across** 5M documents
Compute the word frequency **across** 5M documents

- Distributed list of sets of words
- Workers to aggregate frequency
Compute the word frequency **across** 5M documents
Compute the word frequency **across** 5M documents

A hash function is any function that can be used to map data of arbitrary size to a data of a fixed size.
Compute the word frequency **across** 5M documents.

**Distributed list of sets of words**

**Complex communication phase**

```
reduce ↓ 5
reduce ↓ 4
reduce ↓ 3
```

Workers to aggregate frequency

Now we just count the occurrences per word.
The Map Reduce Abstraction for Distributed Algorithms

Distributed Data Storage

Map

(Shuffle)

Reduce