CSE 5095: Research Topics In Big Data Analytics
Professor: Sanguthevar Rajasekaran
Notes Taken By: William Reynolds
Lecture Date: February 25th, 2014

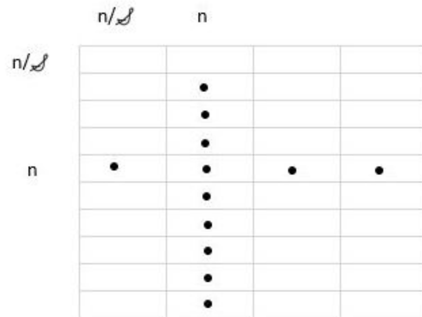

Figure 1: Sorting a mesh with the s^2-way merge sort

In the last lecture we discussed the odd-even merge algorithm and saw how that algorithm can be used to sort n given elements. Given n elements, the idea is to partition the input into two halves, recursively sort each half, and merge the sorted halves using the odd-even merge algorithm. An extension of this algorithm (called the s^2-way merge sort) was proposed by (Thompson and Kung 1977) to sort a mesh. The idea was to partition the mesh into sub meshes of size $\frac{n}{s} \times \frac{n}{s}$, sort each sub mesh, and merge the s^2 sorted sub meshes using the odd-even merge algorithm.

In Figure 1, the sorted subsequences in the mesh are shown as $X_1, X_2, \ldots, X_{s^2}$. Each of these subsequences is partitioned into its odd and even parts; all the odd parts are recursively merged to get Y and all the even parts are merged recursively to get Z; Y and Z are shuffled to get the sequence Q; In the shuffled sequence Q we can show that the length of the dirty sequence is no more than $2s^2$; we clean Q by performing some local sorting.

The algorithm (ℓ, m)-merge sorting (LMM) is an extension of the above algorithms due to (Rajasekaran 1999).

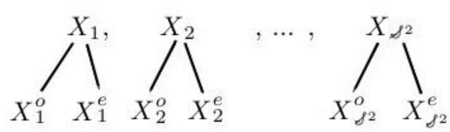

Figure 2: s^2-way merge sort is a special case of the LMM algorithm

$(\ell,\, m)$ Merge Sort (LMM)

Input : $X = k_1, k_2, \ldots, k_n$

Output : Sorted X

Algorithm :

1 Partition X into ℓ equal sized parts: $X_1, X_2, \ldots,$ and X_ℓ.
2 **for** $1 \le i \le \ell$ **do**
    Recursively sort X_i to get Y_i
3 Merge $Y_1, Y_2, \ldots, Y_\ell$ using Algorithm $(\ell,\, m)$ Merge

Input : Sorted sequences $X_1, X_2, \ldots, X_\ell$

Output : Merge of $X_1, X_2, \ldots, X_\ell$

Algorithm :

1. **for** $1 \leq i \leq \ell$ **do**

   Unshuffle X_i into m parts : $X_i^1, X_i^2, \ldots, X_i^m$

   if $X_i = x_i^1, x_i^2, \ldots, x_i^r$

   then $X_i^1 = x_i^1, x_i^{1+m}, x_i^{1+2m}, \cdots$

   $X_i^2 = x_i^2, x_i^{2+m}, x_i^{2+2m}, \ldots$

   .
   .
   .

   $X_i^m = x_i^m, x_i^{2m}, x_i^{3m}, \ldots$

2. **for** $1 \leq i \leq m$ **do**

   Recursively merge $X_1^i, X_2^i, \ldots, X_\ell^i$ to get $Y_i = y_1^i, y_2^i, \ldots$

3. Shuffle $Y_1, Y_2, \ldots, Y_m$

   Let $Y_i = y_i^1, y_i^2, \ldots, y_i^{\ell r/m}$ where $1 \leq i \leq m$

   The shuffled sequence $Z = y_1^1, y_2^1, y_3^1, ..., y_m^1, y_1^2, y_2^2, y_3^2, ..., y_m^2, \ldots, \ldots, y_1^{\ell r/m}, y_2^{\ell r/m}, \ldots, y_m^{\ell r/m}$

Claim : The length of the dirty sequence is no more than ℓm

Let $Z = Z_1, Z_2, Z_3, ...$

for each i where $|Z_i| = \ell m$

4a. Sort and Merge Z_1 & Z_2; Z_3 & Z_4; $\cdots$

4b. Sort and Merge Z_2 & Z_3; Z_4 & Z_5; $\cdots$

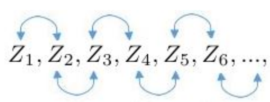

Figure 3: Step 4a (Top Arrows) And Step 4b (Bottom Arrows)

Now we are done!

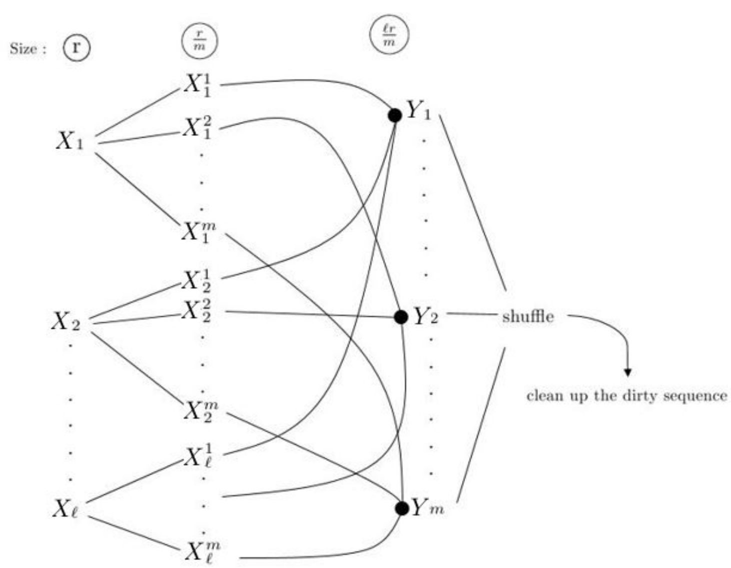

Figure 4: A demonstration of the LMM Algorithm

Proof of Claim :

The minimum number of zeros contributed by any X_i to any $Y_j = \lfloor \frac{n_i}{m} \rfloor$, where n_i is the number of zeros in X_i , $1 \le i \le \ell$

The maximum number of zeros contributed by any X_i to any $Y_j = \lceil \frac{n_i}{m} \rceil$, where $1 \le i \le \ell$ and $1 \le j \le m$

$\Longrightarrow$ The difference between the number of zeros in Y_1 & Y_m is $\le \ell$. As a corollary, it follows that the length of the dirty sequence is no more than ℓm. See Figure 5 for a worst case example. In this example, Y_1 has ℓ more zeros than the others. The other sequences have all ones in these ℓ columns.

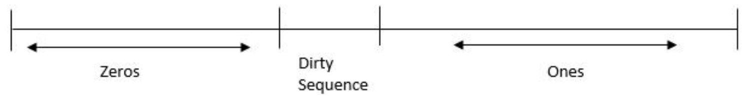

Figure 5: Sequence Z could have a dirty sequence

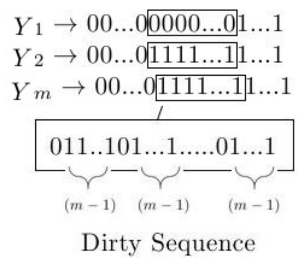

Figure 6: An example input for which the dirty sequence is the longest

An Example : We'll illustrate LMM in sorting $M\sqrt{M}$ elements.

$N = M\sqrt{M}$ $B = D = \sqrt{M}$

Sort N elements

Using LMM we can sort in 3 passes

1 Form runs of length M each; There are $\sqrt{M}$ runs that we have to merge.
Let these runs be $X_1, X_2, \ldots, X_{\sqrt{M}}$

2 Unshuffle each run into $\sqrt{M}$ parts

3 Recursively Merge $X_1^j, X_2^j, ..., X_{\sqrt{M}}^j$ to get Y_j, for $1 \le j \le \sqrt{M}$

4 Shuffle $Y_1, Y_2, \ldots, Y_{\sqrt{M}}$

5 Clean up the dirty sequence

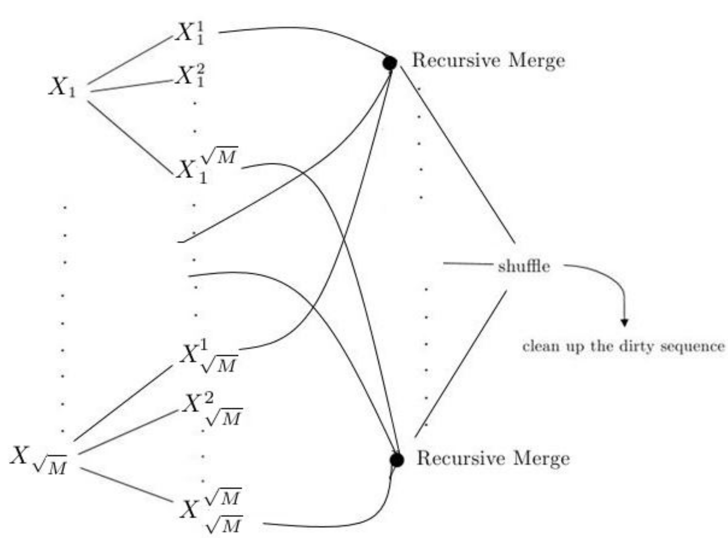

Figure 7: LMM in action for the example sorting problem

Analysis :

- Note that we have used LMM with $\ell = m = \sqrt{M}$. Steps 1 and 2 take 1 pass together.

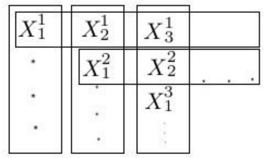

Figure 8: An optimal way to stripe the data after step 2. This striping enables us to perform step 3 in one pass.

- Step 3 takes 1 pass

Assume that we have a memory of size $2M$. In this case we can clean up the dirty sequence while we are shuffling. Let Z be partitioned into blocks of size M each: $Z = Z_1, Z_2, \ldots$, where each block Z_i is of size $\ell m = M$. Note that the dirty sequence can only span two successive blocks. Therefore, one way of cleaning the sequence Z is to: sort and merge Z_1 and Z_2; Z_2 and Z_3; etc. If we have $2M$ memory, we can do this cleaning as well as Step 4 in a total of one pass.

- As a result, Steps 4 and 5 take 1 pass.

$\therefore$ The Total Number of Passes = 3

Note: (Chaudhry and Cormen 2002) have shown that when $B = D = \sqrt{M}$, we can sort $\frac{M\sqrt{M}}{2}$ keys in 3 passes through the data. They have implemented the column sort algorithm of (Leighton 1985). It turns out that the column sort algorithm is indeed a special case of the LMM algorithm. Clearly, odd-even merge sort and the s^2-way merge sort are also special cases of LMM.

Note: When we analyze the I/O complexity of out-of-core algorithms we normally compute only the read complexity. Typically, the write complexity will be similar.