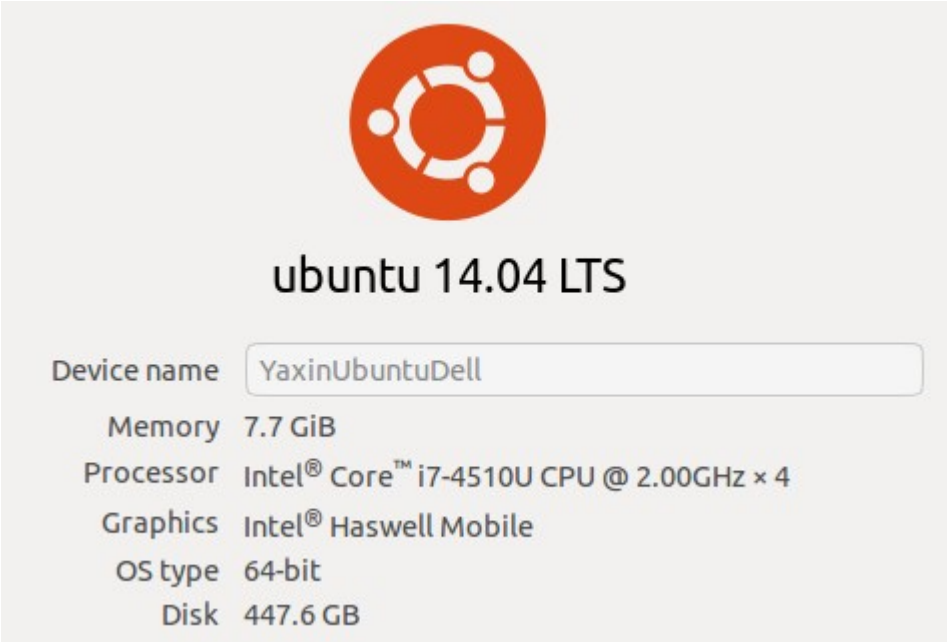
Assignment Report

A2 – Yaxin Huang, Revon Matthews

Created date: Feb 12th, 2015

I. Local Machine Configuration



II. Cluster Configuration

The cluster configuring procedure is the same as the post in the below link:
https://piazza.com/class/i4ppp5auv1n7k5?cid=94

And the instances running are:

Name	Status	Type	Instance Type	Count
Core instance group - 2	Running	CORE	m1.small	2 Resize
Master instance group - 1	Running	MASTER	m1.small	1

The specification of m1.small is as below:

Instance Family	Instance Type	Processor Arch	vCPU	Memory (GiB)	Instance Storage (GB)	EBS-optimized Available	Network Performance
General Purpose	m1.small	32-bit or 64-bit	1	1.7	1 x 160	-	Low

III. Running the Original Version

The original version is the same definition as the v2 from A1. It reads the input file and put it to the Mapper directly, and no combiner is used. The original version is the one in folder “A2_original”.

1. Running the Original Version in Standalone Mode

The running time and output are given here to help compare the original version to the one with bin-size and sample-rate.

(a) Running time summary

Because of time constraint, the code is only run on my machine three times.

1St run: 117.97 s
2nd run: 121.97 s
3rd run: 115.71 s
Average running time: 118.55 s

(b) Output

The output can be found in the “OutputRecords.ods” file.

2. Running the Original Version in Cluster

The running time and output are given here to help compare the original version to the one with bin-size and sample-rate.

(a) Running time summary

Because of time constraint, the code is only run on my machine three times.

1St run: 527.43 s
2nd run: 552.73 s
3rd run: 569.298 s
Average running time: 549.82 s

(b) Output

The output can be found in the “OutputRecords.ods” file.

IV. Running the Revised Version

1. Standalone Mode

(a) Running time comparison:

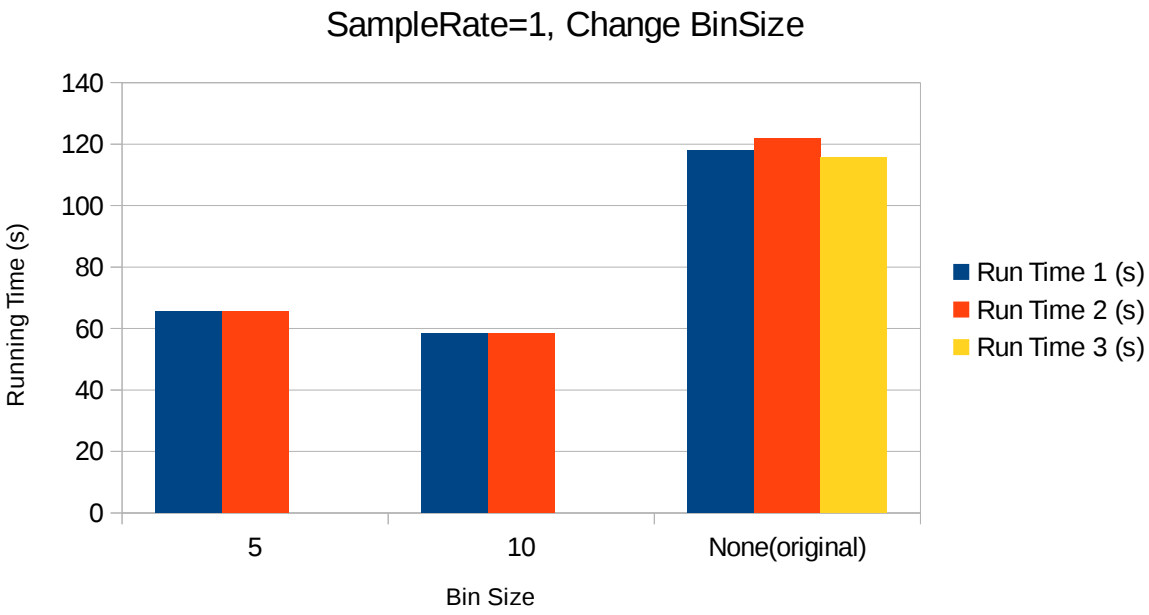


Figure-1 SampleRate=1, Chaning the BinSize

The above figure-1 shows the performance of different runs when we didn't decrease the amount of data sent to map and kept changing the bin size. Because of time constraint we only tested on binsize = 5, 10 and a limited number of runs. But still you can see that the running time goes down as the bin size increases. (But I think when bin size increases to a certain number, the running time will start to go up.)

For example, if bin size = 5, in the Combiner for every 5 price numbers in a category, the Combiner will sort the 5 numbers and returns an intermediate medium.

The reason why a good bin size can decrease the running time:
Suppose the sorting time takes O(nlogn), and n is the length of the price list.
With no combiner:
Time(original) = Time to sort whole list + Else
= O(nlogn) + Else
With bin size = 5:
Time(bin5) = (Time to sort 5 numbers) * n/5 + (Time to sort n/5 numbers) + Else
= O(nlog5) + O(n/5 * (logn) – log5) + Else
= O((4n/5) * log5 + (n/5)logn) + Else < Time(original)

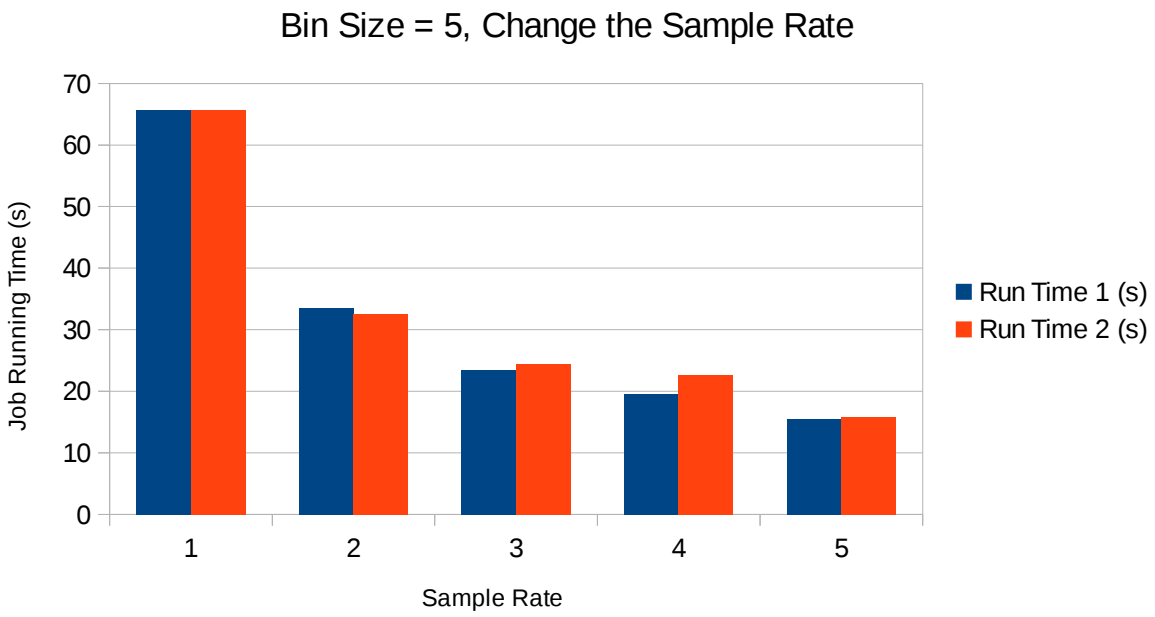


Figure-2 Bin size = 5 (except the original one), changing the sample rate

Figure-2 shows when we keep bin-size fixed, what's the job running time using different sample rate. The sample rate here is always an integer larger than 0. When sample rate = 1, every lines in the input file will be sampled. When sample rate = x > 1, the first line of every x lines will be sampled.

As we can see the job running time decreases as the sample rate increases.

(b) Different bin-size & sample-rate combinations, the corresponding running time and variance of medians:

In the following table, the performance of different bin sizes and sample rates. Here we measure the sampling time, job running time and the variance, where the variance is calculated by:

$$\text{Variance(bin-size, sample-rate)} = \text{sum[for all categories } (\text{Median}_{output} - \text{Median}_{original\_output})^2] \text{ / number\_of\_categories}$$

since for certain bin-size & sample-rate pair, the output will always be the same in our implementation.

Table-1 Standalone Mode Performance Difference

Bin Size	**Sample Rate**	**Average Sampling Time(s)**	**Average Job Running Time(s)**	**Variance (5 digits after decimal point)**
Original	Original	N/A	118.55	0.00000
5	1	56.60	65.65	0.07112
5	2	47.50	33.03	0.26916
5	3	43.72	23.95	0.57500
5	4	49.09	21.04	0.82051
5	**5**	**42.56**	**15.68**	**0.09706**
10	1	52.53	58.55	0.00834
10	2	47.75	31.58	0.02835
10	3	43.93	21.50	0.46222
10	4	44.25	16.51	1.02462
10	**5**	**43.98**	**13.52**	**0.10519**
15	1	48.97	56.81	0.02936
20	1	53.72	56.16	0.01858
25	1	49.25	55.65	0.03462

As table-1 shows, although we decrease the number of samples sent to Mapper and use Combiner to compute the intermediate medians, the result is still good as the variance is acceptable. The bold lines are when sample rate is 5, the job running time is cut down to almost one tenth, but the performance is still good. (This is probably because the input file is randomly generated using Normal Distribution, which causes most of the prices fall among 250.)

2. Cluster Mode: Changing the Bin Size

In cluster mode, out of some mysterious reason, our sampling code is buggy and always give “FileCannotBeFound” Exception where we need to create a sampled file in line 86 of MedianOfPurchases.java. Thus, we remove the sampling function and only the bin size is used in cluster mode.

Number of reducers in the cluster mode is automatically set to 3, since there are always 3 output files generated.

Table-2 Cluster Mode: Changing the Bin Size

Bin Size	Average Job Running Time(s)	Variance (5 digits after decimal point)
5	452.06	0.07641
10	456.62	0.07641
15	450.83	0.07641
20	460.04	0.07641
25	461.28	0.07641

When I inspected the output I found that the output were all the same so they have the same variance. The output medians are different from the ones from original version, which indicates that the Combiner was used. However, table-2 shows that the combiner didn't help a lot and the average running time was even increasing as the bin size increased.