

# **Efficient Algorithms for Large-Scale Temporal Aggregation**

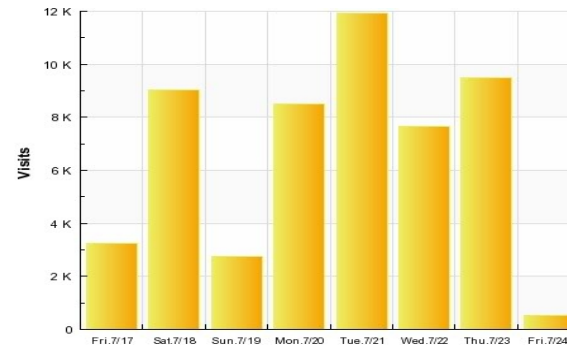
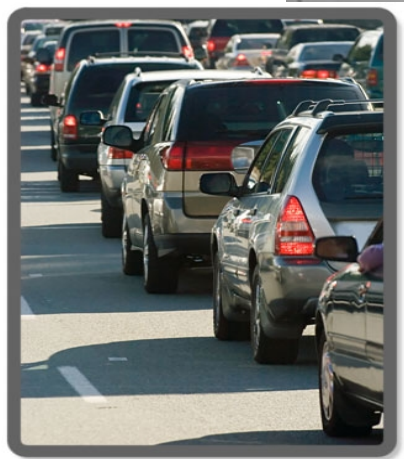
Bongki Moon, Ines Fernando Vega Lopez, and  
Vijaykumar Immanuel

presented by

Nihal Dindar

# Motivation

- Databases need to capture and query time-varying nature of the data for many applications such as



| Day of the Week | Visits |
|-----------------|--------|
| Fri, 7/24       | 521    |
| Thu, 7/23       | 9,510  |
| Wed, 7/22       | 7,660  |
| Tue, 7/21       | 11,915 |
| Mon, 7/20       | 8,520  |
| Sun, 7/19       | 2,766  |
| Sat, 7/18       | 9,031  |
| Fri, 7/17       | 3,259  |



# Talk Outline

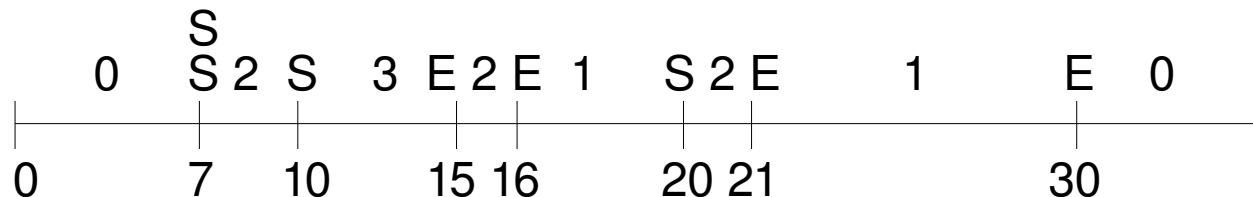
- Problem
- Approach
- Solution
  - Sequential Algorithm
    - Small-scale Aggregation
    - Large-scale Aggregation
  - Parallel Algorithm
- Evaluation
- Conclusion

# Problem: Varying Interval Size

- Stock Exchange Example:

| Symbol    | Price | Volume | Begin | End |
|-----------|-------|--------|-------|-----|
| IBM       | 37    | 1000   | 10    | 30  |
| Microsoft | 45    | 2000   | 7     | 16  |
| Google    | 30    | 2000   | 7     | 15  |
| HP        | 32    | 2000   | 20    | 21  |

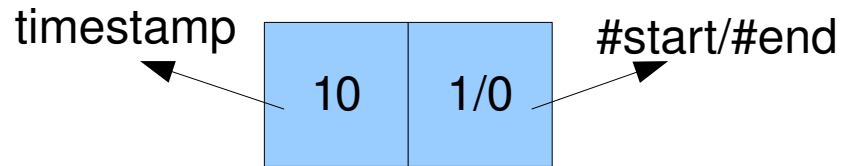
- Query: compute the number of trades at a given time interval
- Naïve Solution
  - Sorted list of timestamps, intermediate counts



# Approach

- Small Scale Aggregation
  - Assumption: Large enough memory to store the entire data structure
  - Group Aggregation Operator into 2 based on the need to keep track of individual tuples
    - Group 1: **Count**, Sum, Avg
    - Group 2: **Max**, Min
- Large Scale Aggregation
  - Constraint of limited buffer space
- Scalability with the increasing number of processors

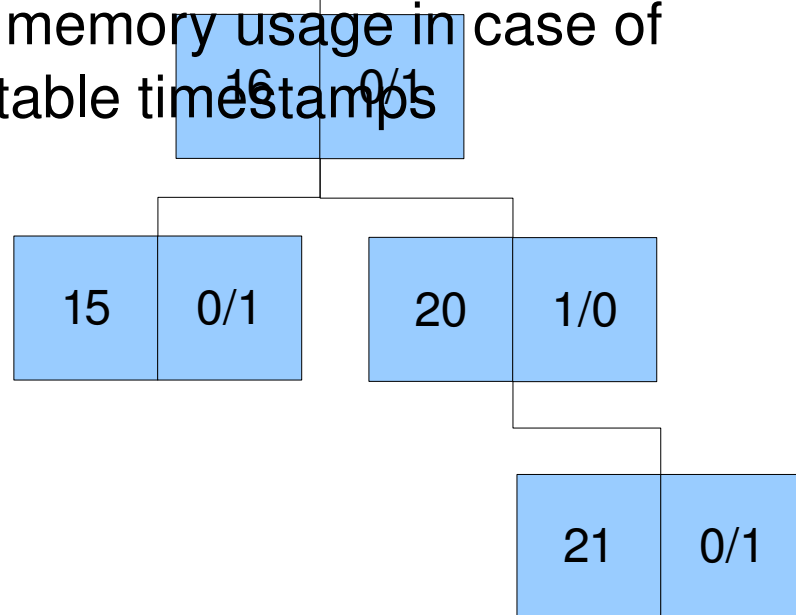
# Small-Scale Aggregation: Balanced Tree



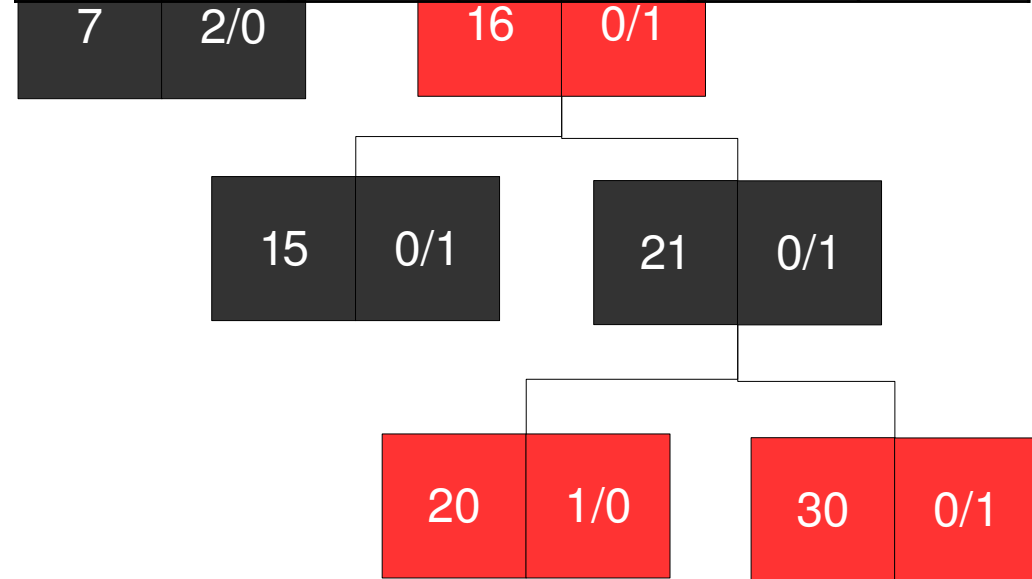
- Incrementally sorting the data



- Less memory usage in case of repeatable timestamps

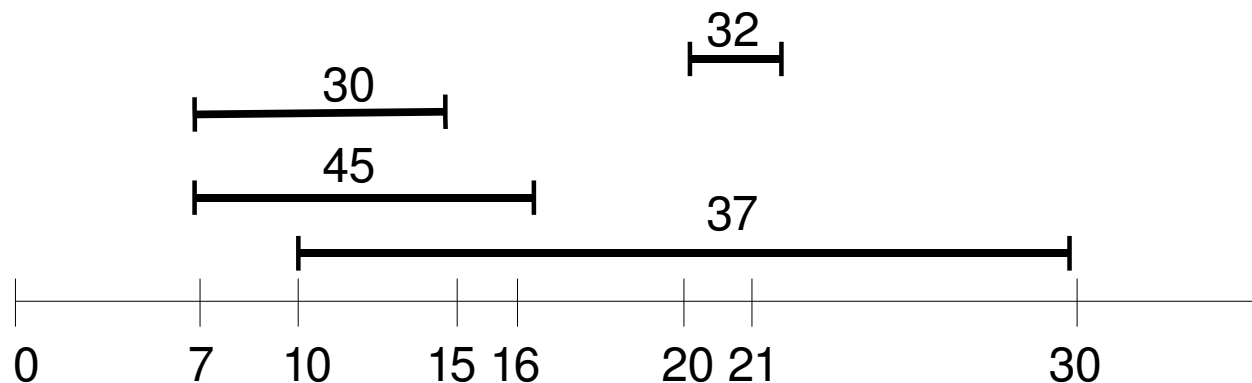


| Symbol    | Price | Volume | Begin | End |
|-----------|-------|--------|-------|-----|
| IBM       | 10    | 1000   | 10    | 30  |
| Microsoft | 45    | 2000   | 7     | 16  |
| Google    | 30    | 2000   | 7     | 15  |
| HP        | 32    | 2000   | 20    | 21  |



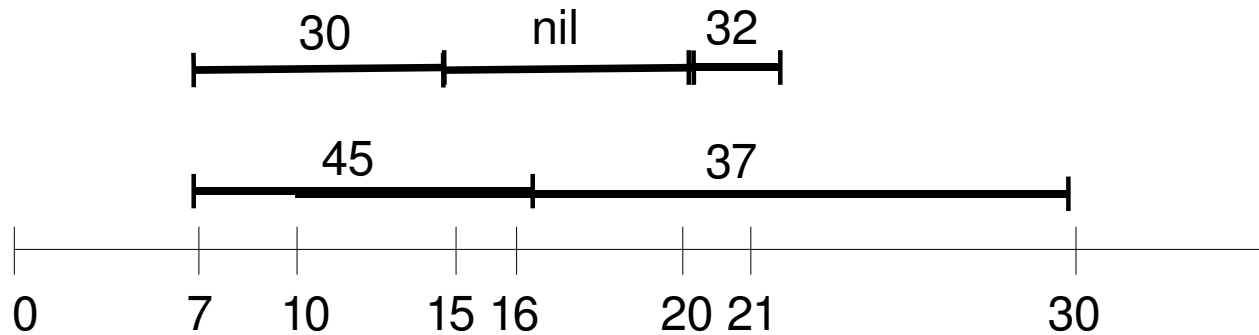
# Small-Scale Aggregation(cont): Merge Sort Aggregation

- Similar to the classical merge-sort algorithm based on divide and conquer
- Stock Exchange Example: Maximum Price
- Step 0:

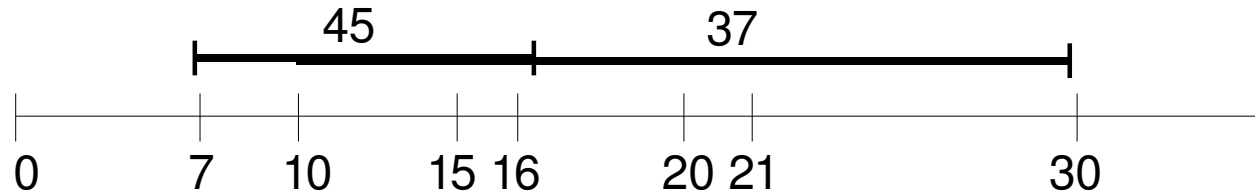


# Merge Sort Aggregation (Cont)

- Step 1:



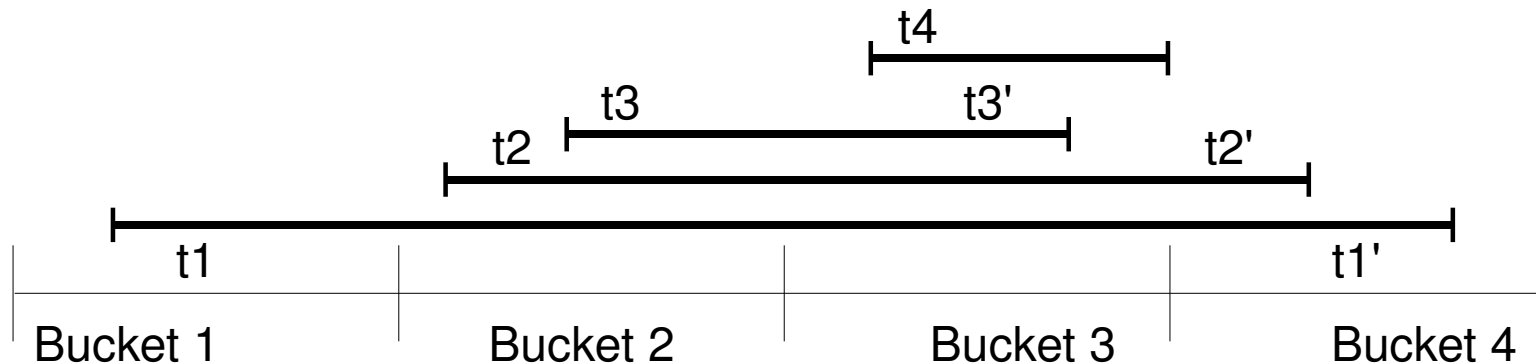
- Step 2:



- Worst case performance is  $O(N \log N)$

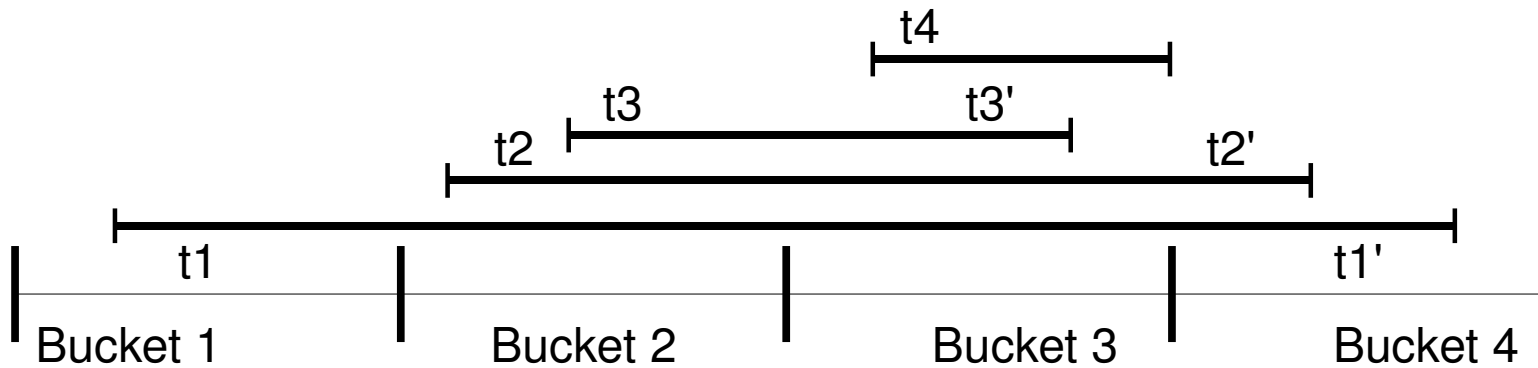
# Large-Scale Aggregation: Bucket Algorithm

- Time interval is divided into N number of disjoint intervals
- Each bucket stores the tuples which have started or ended at its interval
- Meta Array is introduced for long-lived tuples

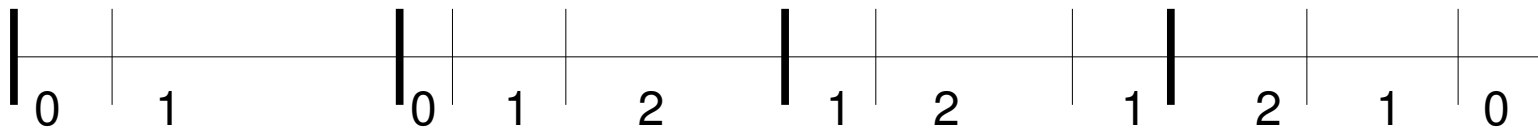


Bucket 1 = {t1}      Bucket 3 = {t4, t3'}      Meta Array for count aggregation = {0,1,2,0}  
Bucket 2 = {t2, t3}      Bucket 4 = {t2', t1'}

# Bucket Algorithm(Cont)



Bucket 1 = {t1}      Bucket 3 = {t4, t3'}  
 Bucket 2 = {t2, t3}      Bucket 4 = {t2', t1'}

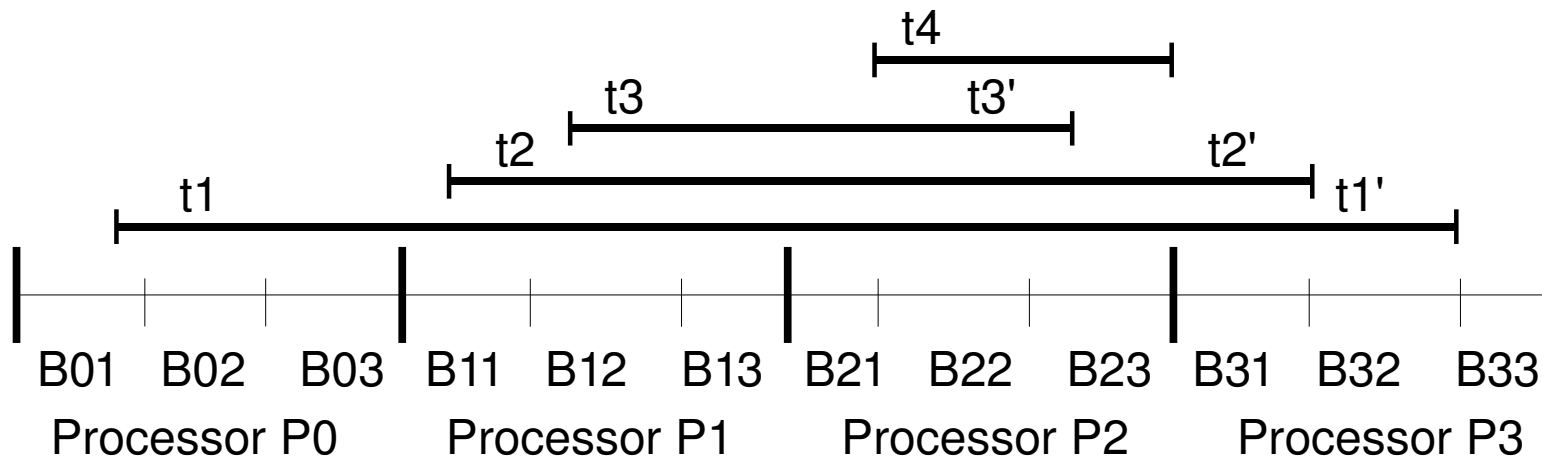


Meta Array for Count Aggregation = {0,1,2,0}



- Performance depends on the number of buckets

# Parallel Bucket Algorithm



**Bucket B01 = { $t1$ }**

**Bucket B02 = {}**

**Bucket B03 = {}**

**P0's Local Meta Array for count aggregation = {0,1,1}**

**P0's Global Meta Array for count aggregation = {0,1,2,0}**

- Similar to Bucket Algorithm, the important difference is the cost of communication among different processors to compute the final

# Evaluation

- Balanced Tree & Merge Sort Aggregation Algorithms
  - small-scale aggregation algorithms improved worst case and average case processing time of an aggregation tree algorithm
- Bucket Algorithm
  - can compute the temporal aggregates for databases larger than the size of available memory
- Parallel Bucket Algorithm
  - scales up linearly with respect to the increasing number of processors

# Conclusion

- Paper proposed several algorithms to processes temporal aggregations more efficiently
- Special algorithms are designed for different scenarios such as different size of data and operators
- Scalability is achieved with parallelization