Database Group

Architecture for Stream OLAP Exploiting SPE and OLAP Engine

Explosive increase of real-time data streams gives highly demands for real-time analysis over the streams. However, developing tailor-made systems for such applications is not always desirable due to high developing costs. To cope with this problem, we propose a novel architecture for online analytical processing (OLAP) over streams exploiting off-the-shelf stream processing engine (SPE) combined with OLAP engine. It allows users to perform OLAP analysis over streams for the latest time period, called Interval of Interest (IoI). The system in the meantime processes multiple continuous query language (CQL) queries corresponding to different aggregation levels in cube lattice. To cover arbitrary aggregation levels using limited system's memory, we propose to partially deploy CQL queries for those with higher reference frequencies, whereas the results are dynamically calculated using existing aggregation results with the help of OLAP engine. For optimal CQL query deployment, we propose a cost-based optimization method that maximizes the performance. The experimental results show that the proposed system significantly outperforms other comparative methods.

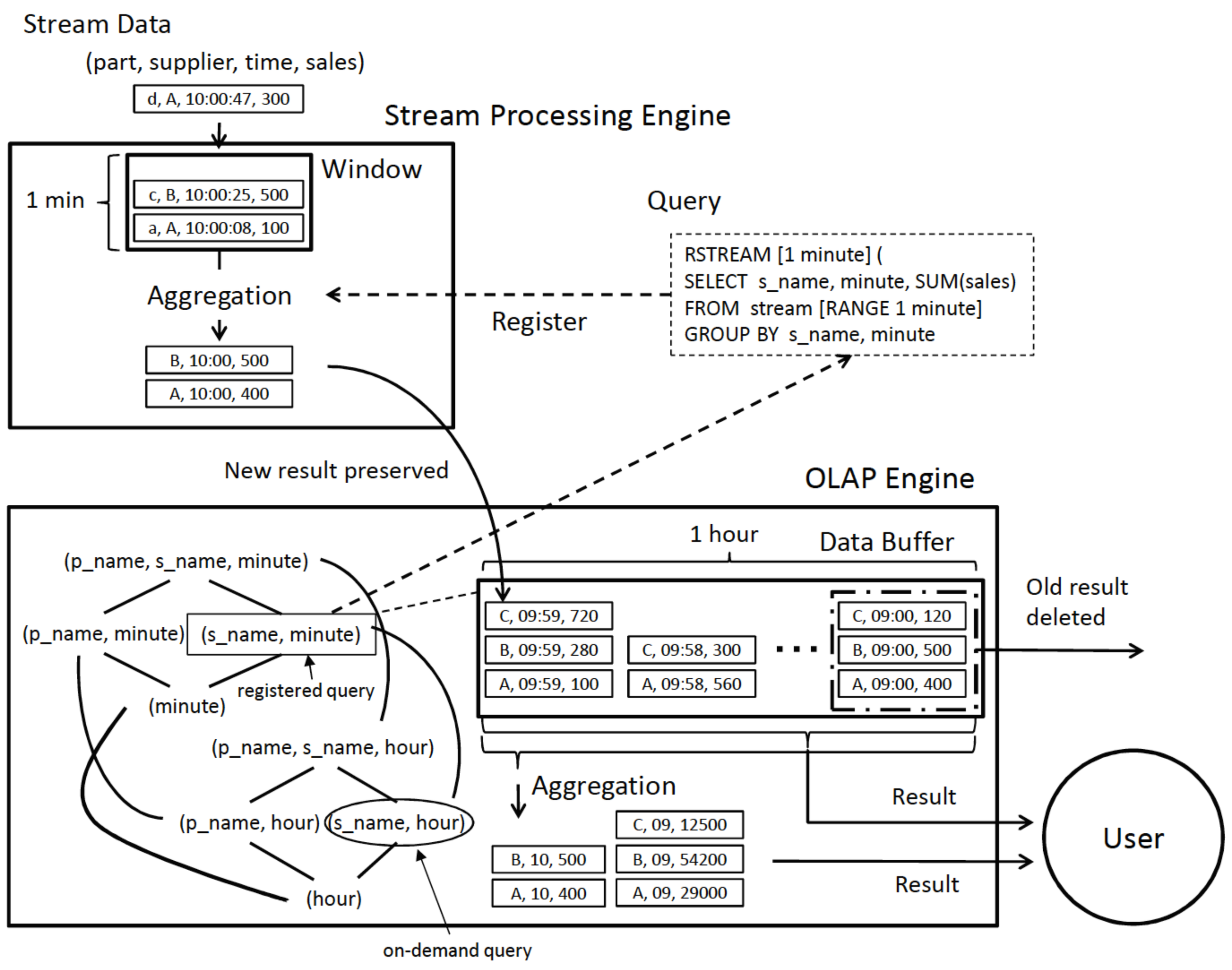


Fig. 1: Architecture of Stream OLAP system

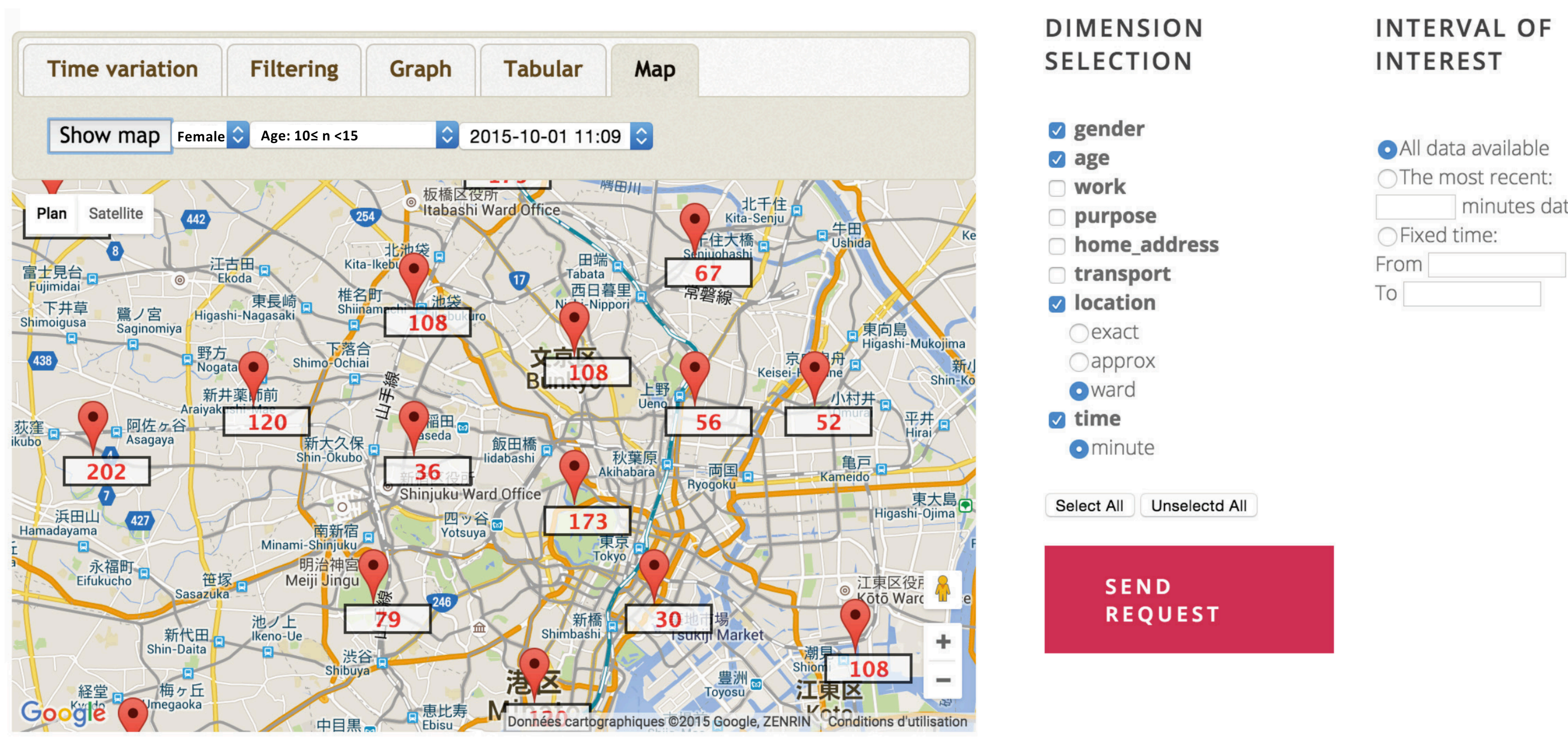


Fig. 2: Stream OLAP system analyzing "People Flow Data"

Parallel Canopy Clustering on GPUs

Canopy clustering is a preprocessing method for standard clustering algorithms such as k-means and hierarchical agglomerative clustering. Canopy clustering can greatly reduce the computational cost of clustering algorithms. However, canopy clustering itself may also take a vast amount of time for handling massive data, if we naively implement it. To address this problem, we present efficient algorithms and implementations of canopy clustering on GPUs, which have evolved recently as general-purpose many-core processors. We not only accelerate the computation of original canopy clustering, but also propose an algorithm using grid index. This algorithm partitions the data into cells to reduce redundant computations and, at the same time, to exploit the parallelism of GPUs. Experiments show that the proposed implementations on the GPU is 2 times faster on average than multi-threaded, SIMD implementations on two octa-core CPUs.

Algorithm 1: Simple Canopy Clustering.

Input: A set S of data points $\boldsymbol{x}_i$, thresholds T_1 and T_2

$\mathcal{C} \leftarrow \emptyset$ ▷ $\mathcal{C}$ is a set of canopies
$\Sigma \leftarrow S$ ▷ Σ is a set of center candidates
while $\Sigma \neq \emptyset$ **do**
  $\boldsymbol{c} \leftarrow$ get a point from Σ at random ▷ $\boldsymbol{c}$ is a center
  $C \leftarrow \emptyset$
  for $\boldsymbol{x} \in S$ **do**
    if $d(\boldsymbol{x}, \boldsymbol{c}) \leq T_1$ **then**
      $C \leftarrow C \cup \{\boldsymbol{x}\}$ ▷ a canopy C includes a point $\boldsymbol{x}$
    end
    if $d(\boldsymbol{x}, \boldsymbol{c}) \leq T_2$ **then**
      $\Sigma \leftarrow \Sigma - \{\boldsymbol{x}\}$ ▷ remove $\boldsymbol{x}$ from the candidates
    end
  end
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$
end
return $\mathcal{C}$

```
while there exist candidates do
    compute_distances();
    create_canopy();
    select_center();
end
```

(1)Divide while loop into 3kernel functions
(2)Run the functions on GPUs

Fig. 3: Canopy Clustering Algorithm on GPUs

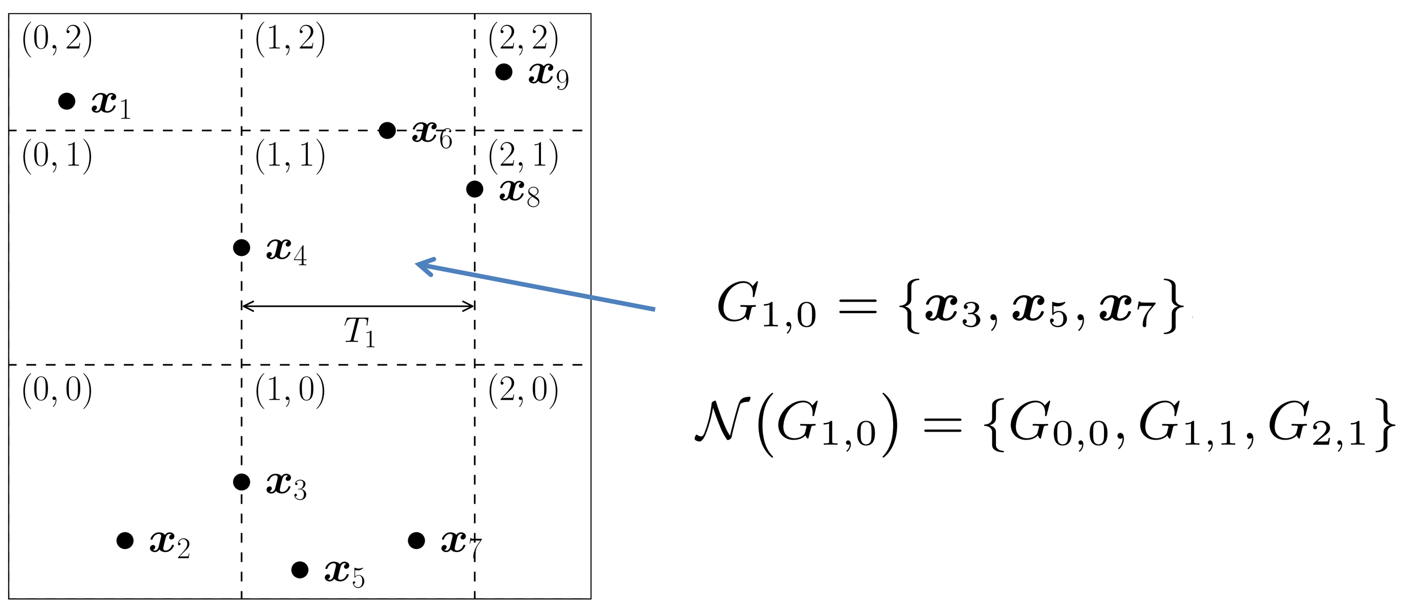
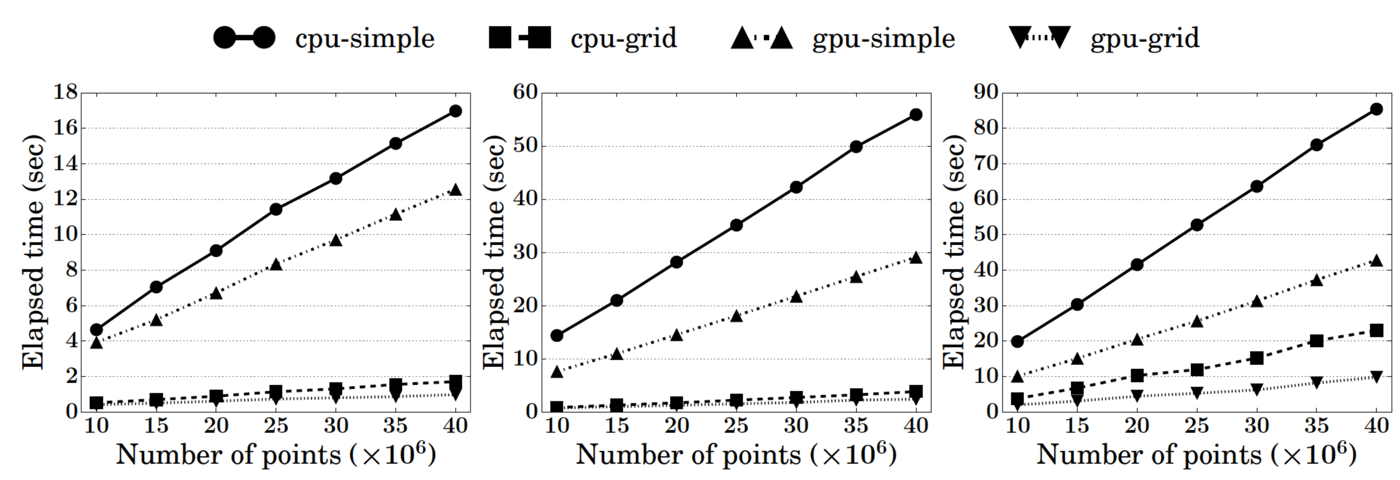


Fig. 4: An example of grid index



(a) $d = 2, T_1 = 0.02$. (b) $d = 4, T_1 = 0.15$. (c) $d = 6, T_1 = 0.35$.

Fig. 5: Overall performances (d: dimensionality, T_1: grid size)

contact address: {kitagawa, amagasa, shiokawa}@cs.tsukuba.ac.jp