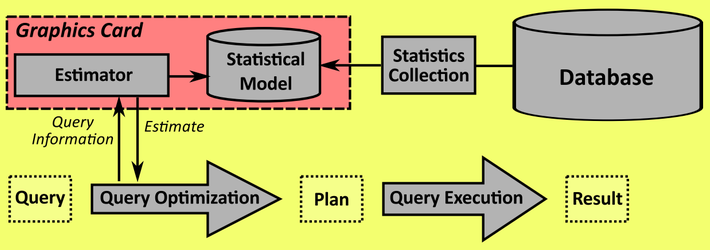
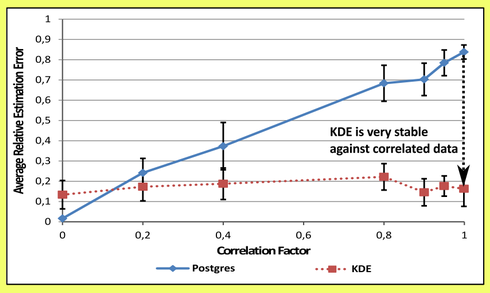
Demonstrating Transfer-Efficient Sample Maintenance on Graphics Cards

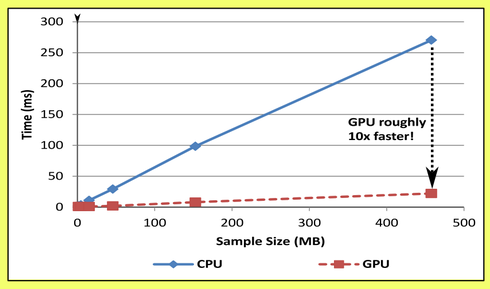
Martin Kiefer, Max Heimel, Volker Markl

Setting: GPU-assisted Selectivity Estimation

- **Idea: Use Graphics Card as a "statistical co-processor" during query optimization.**
  - *Keep statistical model of the data on the GPU.*
  - *Compute selectivity estimates based on this model.*







- **Enables larger models & more complex methods.**
  - *Produces better estimates, resulting in better query plans!*
  - *Indirect method of using GPUs to accelerate database systems that does not suffer from data transfer / data placement / etc...*

Heimel et.al. - A first step towards GPU-assisted Query Optimization. ADMS 2013

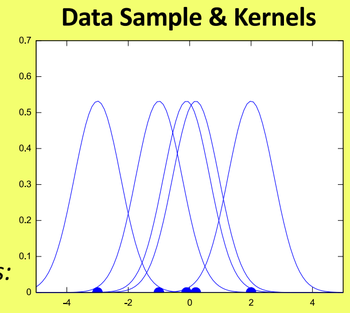
- **We use Kernel Density Estimation (KDE):**
  - *State-of-the art estimation method from statistics literature.*
  - *Converges faster to true distribution than MD-histograms.*
  - *Computes selectivities based on a data sample.*
  - ***Step 1:** Draw a uniform sample $S = \{\vec{t}^{(1)}, \vec{t}^{(2)}, ..., \vec{t}^{(s)}\} \subseteq R$.*
  - ***Step 2:** Center Kernel functions K_H around sample points.*
    - Kernel functions are basically symmetric probability density functions.
  - ***Step 3:** Estimate the density at a point by averaging the Kernels:*

$$\hat{p}_H(\vec{x}) = \frac{1}{s}\sum_{i=1}^{s} K_H(\vec{t}^{(i)} - \vec{x})$$

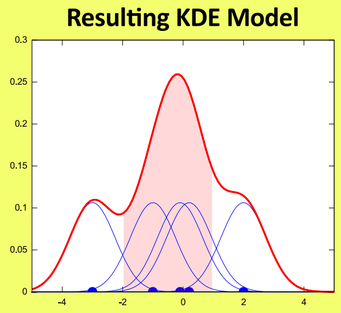
  - ***Step 4:** Estimate selectivity for a region Ω by integration:*

$$\hat{p}_H(\Omega) = \int_\Omega \hat{p}_H(\vec{x})d\vec{x} = \frac{1}{s}\sum_{i=1}^{s} \underbrace{\int_\Omega K_H(\vec{t}^{(i)} - \vec{x})}_{\hat{p}_H^{(i)}(\Omega)}$$

Data Sample & Kernels



Resulting KDE Model

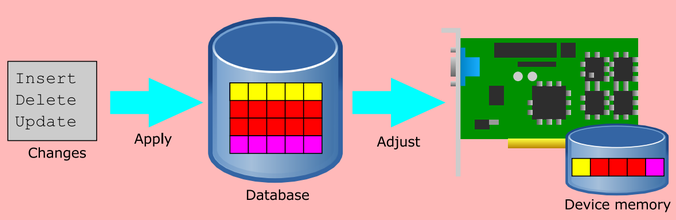


- **Produces high-quality estimates, but is expensive!**
  - *Luckily, it can be efficiently parallelized and mapped to a GPU!*
  - ***Using a GPU enables a superior selectivity estimation method!***

Heimel et.al. - Self-Tuning, GPU-Accelerated Kernel Density Models for Multidimensional Selectivity Estimation. SIGMOD 2015

Problem: KDE Model Maintenance

- **What happens when the database changes?**
  - *KDE estimates are computed from a data sample.*
  - *If the user changes data, the sample becomes stale, resulting in incorrect estimates & degraded plan quality.*
  - ***We need to maintain the sample wrt database updates!***



- **Sample maintenance is a well-known problem. However: Our sample is located on a graphics card!**
  - *Maintenance has to be applied over the (limited) PCI Express bus.*
  - *We have to make sure that we do not saturate the bus!*
  - ***The maintenance algorithm must be transfer-efficient!***
    - Good sample quality, while being economical with data transfers.

Correlated Acceptance/Rejection (CAR)

- **State-of-the-art sample maintenance algorithm, dealing with insertions, updates and deletions.**
- **Is CAR a transfer-efficient algorithm?**

```
on_insertion(inserted_t){
    r = draw from BINOM(|S|, 1/|R|)
    remove r random elements from S
    fill S with instances of inserted_t
}

on_update(before_t, after_t){
    replace before_t in S with after_t
}

on_deletion(deleted_t){
    remove instances of deleted_t in S
    fill S with sampled tuples from S
}
```

Olken et.al. - Maintenance of materialized views of sampling queries. ICDE 1992

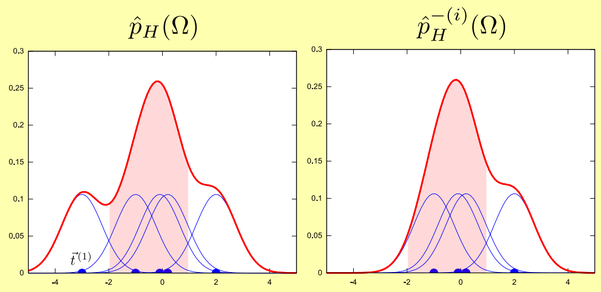
- **Insertions:** ✓
  - *Similar to Reservoir Sampling.*
  - *Only requires a transfer if sample is actually affected!*
  - *All relevant decision can be made independently by the host.*
- **Updates:** !
  - *Two tuples have to be transferred for **every** update.*
- **Deletions:** !
  - *The deleted tuple has to be transferred for **every** deletion.*
  - *A map identifiying deleted tuples has to be transferred to main memory for **every** deletion.*
  - *The sampled tuples have to be transferred.*
- **Can we be more transfer-efficient than CAR for update-intensive workloads by dropping uniformity?**

The Karma Score: Error-driven sample maintenance

- **Key facts:**
  - *We know the query's selectivity $p(\Omega)$ after execution.*
  - *In KDE, every sample point contributes probability mass independent of the others.*
- **We can quantify the impact on the estimation error for every sample point after the query!**
  - *Compute the "adjusted estimate" for every point by removing the point's contribution:*

$$\hat{p}_H^{-(i)}(\Omega) = \frac{\hat{p}_H(\Omega)\cdot s - \hat{p}_H^{(i)}(\Omega)}{s-1}$$

  - *For "bad" points, this adjusted estimate should be closer to the true selectivity. Vice versa for "good" points.*
  - *Gives an inidcation whether points are "helpful".*



- **Karma Score: By how much has the point changed the estimation error?**

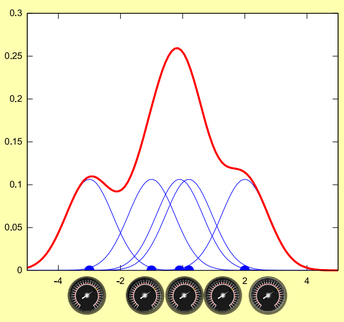
$$K^{(i)}(\Omega) = |S|\cdot\left(\mathcal{L}_{abs}(p(\Omega), \hat{p}_H(\Omega)) - \mathcal{L}_{abs}\left(p(\Omega), \hat{p}_H^{-(i)}(\Omega)\right)\right)$$

  - *Good contribution --> Positive Karma.*
  - *Bad contribution --> Negative Karma.*

- **We aggregate the Karma scores over subsequent queries.**

$$K_{t+1}^{(i)} = \min\left(K_t^{(i)} + K^{(i)}(\Omega), K_{\max}\right)$$

  - *Gives an indication how "valuable" a sample point is overall.*
  - *Cumulative Karma is limited to $K_{\max}$ to accelerate sample convergence.*



- **GPU implementation:**
  - *Karma can be calculated on the GPU by one addtional embarassingly parallel computation.*
  - *We need one additional array on device memory to store the aggregated Karma.*
- **We use the cumulative Karma of sample points as a foundation for heuristic sample maintenance algorithms.**

Periodic Karma Replacement (PKR)

- **Every n queries, replace point with worst Karma.**

```
on_query(estimated_sel, true_sel){
    update_karma(estimated_sel, true_sel)
    every k-th query
        resample point with worst karma
}
```

- **GPU implementation:**
  - *A parallel reduction is run to compute the minimum.*
  - *Position of worst point is transferred to host.*
  - *A newly sampled point is transferred back to replace it.*

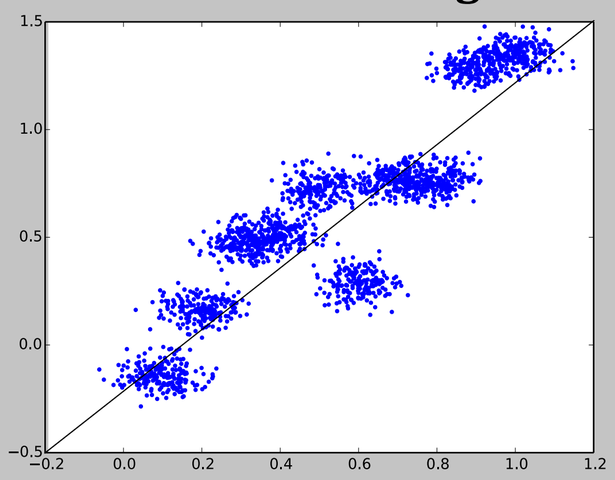
Triggered Karma Replacement (TKR)

- **Replace points whose Karma is worse than θ.**

```
on_query(estimated_sel, true_sel){
    update_karma(estimated_sel, true_sel)
    resample points with karma worse than θ
}
```

- **GPU implementation:**
  - *Offending sample points are computed in parallel.*
  - *Bitmap identifying outdated points is transferred to host.*
  - *Newly sampled points are transferred to device.*

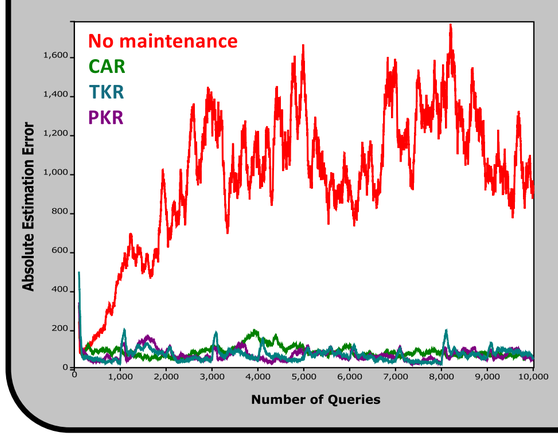
Demo workload: Moving clusters



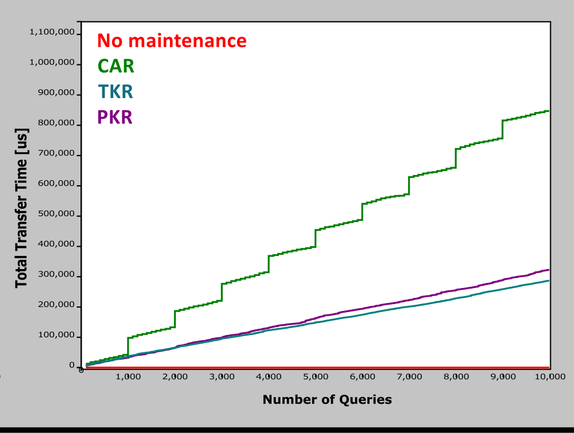
- **Clusters are created along a line.**
- **Moving query target, changing data:**
  - *We start with three clusters.*
  - *Clusters are queried while the next cluster is inserted.*
  - *Periodically, the oldest cluster is deleted.*
  - *Query focus shifts towards the most recent cluster.*
  - *Ten clusters are created and deleted at runtime.*
- **Workload options**
  - *Points per cluster*
  - *Cluster spread*
  - *Algorithm hyperparameters*
  - *Point / query ratio*
  - *Dimensions*
- **Models evolving database with an archive.**

Demo results

- **Estimation error over time:**



- **Cumulative transfer time:**



- **Conclusion:**
  - *GPU-assisted selectivity estimation requires sample maintenance!*
  - *Karma-Based Sample Maintenance can drastically reduce data transfers while offering quality comparable to CAR.*
  - *Generally, TKR is preferrable over PKR (better quality, more reactive).*
  - *However: Depending on workload properties, CAR is still preferrable.*

Source code is available at:
http://goo.gl/aQSQNd