

# Adaptively Parallelizing Distributed Range Queries

Ymir Vigfusson, Adam Silberstein, Brain F. Cooper,  
Rodrigo Fonseca

VLDB '09

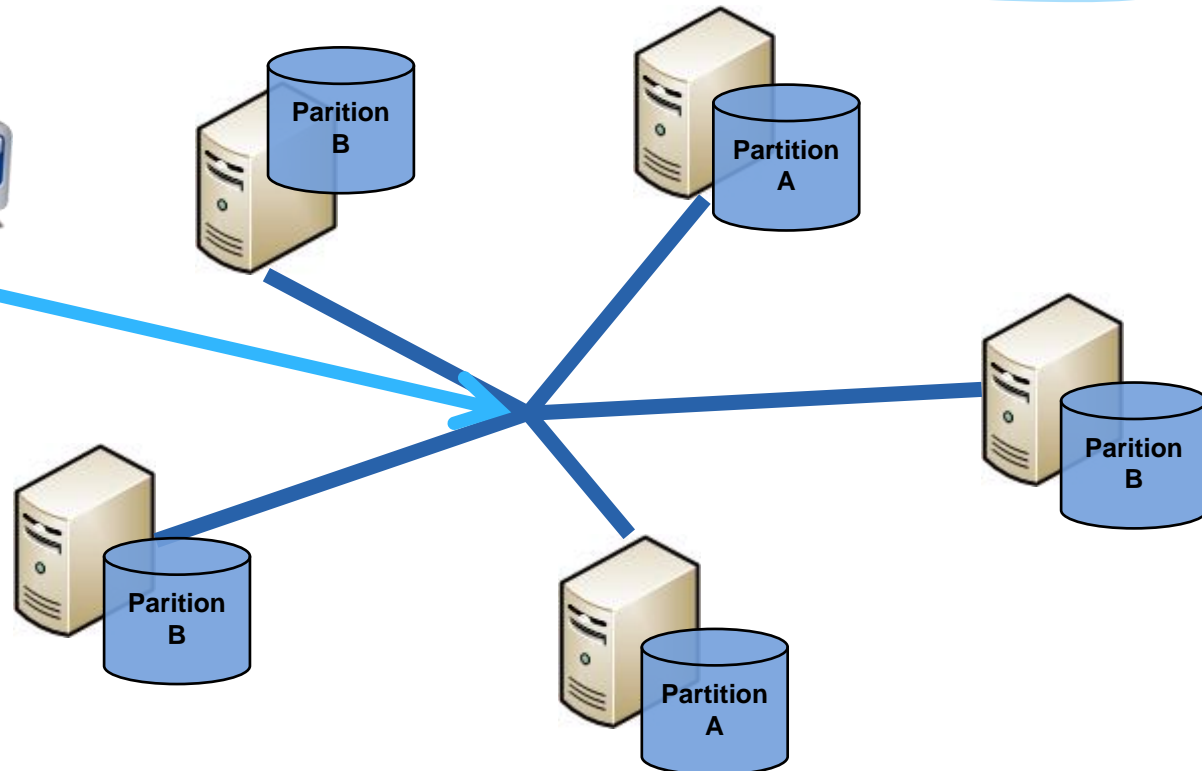
Presented by Anjo Vahldiek



# Environment



# Environment



# Motivation



- \* What is the right **level of parallelism**?
  - \* Parallelism too high → Waste of Performance
  - \* Parallelism too low → Slow acting Service
- \* How do we **assign** a query to the servers?
  - \* Which replica should execute one of the tasks?
- Difficult to optimize a priori

# Agenda



- \* Engineering a new Database
  - \* Problem Description & Requirements
  - \* PNUTS: System Design & Architecture
- \* Optimize Parallelism
  - \* Adaptive Server Allocation Algorithm
  - \* Server Scheduling
- \* Evaluation

# Agenda



- \* Engineering a new Database
  - \* Problem Description & Requirements
  - \* PNUTS: System Design & Architecture
- \* Optimize Parallelism
  - \* Adaptive Server Allocation Algorithm
  - \* Server Scheduling
- \* Evaluation



# Problem Description



- \* **Range Queries over ordered data**
  - \* Time-ordered data
  - \* Secondary indexes
  - \* Hierarchical clustering
- \* **Example:**  
List all new products (within last 24h) under 10€ in Saarland.

# Requirements



- \* General distributed systems requirements (fault-tolerance, availability, scalability, geographic scope, response time, ...)
- \* **Statistics-Free**
- \* Many concurrent clients
- \* Variety of request characteristics
- \* Clients with varying capabilities

# Evaluation Criterion



- \* Time until **first result** (latency)
  - \* Stream of results
- \* Time to finish the whole query (run time)

# Agenda



- \* Engineering a new Database
  - \* Problem Description & Requirements
  - \* PNUTS: System Design & Architecture
- \* Optimize Parallelism
  - \* Adaptive Server Allocation Algorithm
  - \* Server Scheduling
- \* Evaluation



# PNUTS: System Design



- \* Functions (API):
  - \* Get
    - \* Find record of primary key
  - \* Set
    - \* Set record of the primary key
  - \* Delete
    - \* Delete the record defined by the primray key
  - \* **Getrange**
    - \* Retrieve all records in primary key range passing an (optional) predicate
- \* **Per-record consistency**

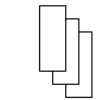
# PNUTS: Architecture



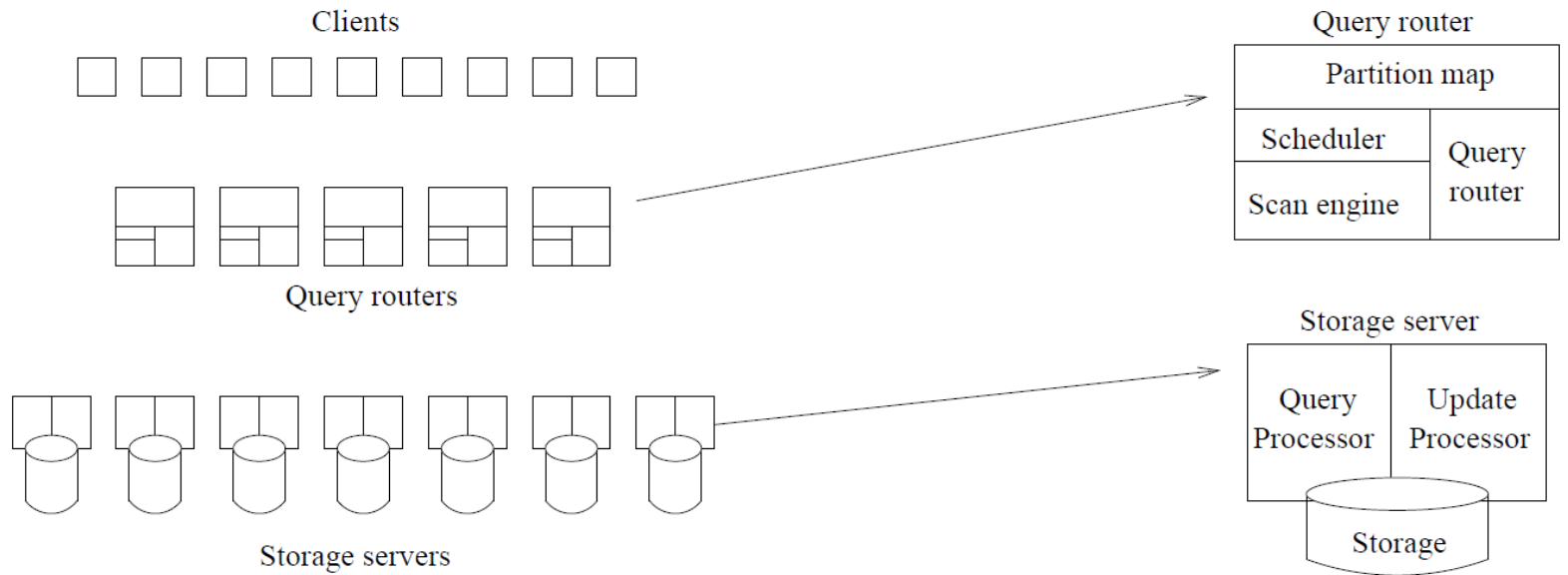
UNIVERSITÄT  
DES  
SAARLANDES



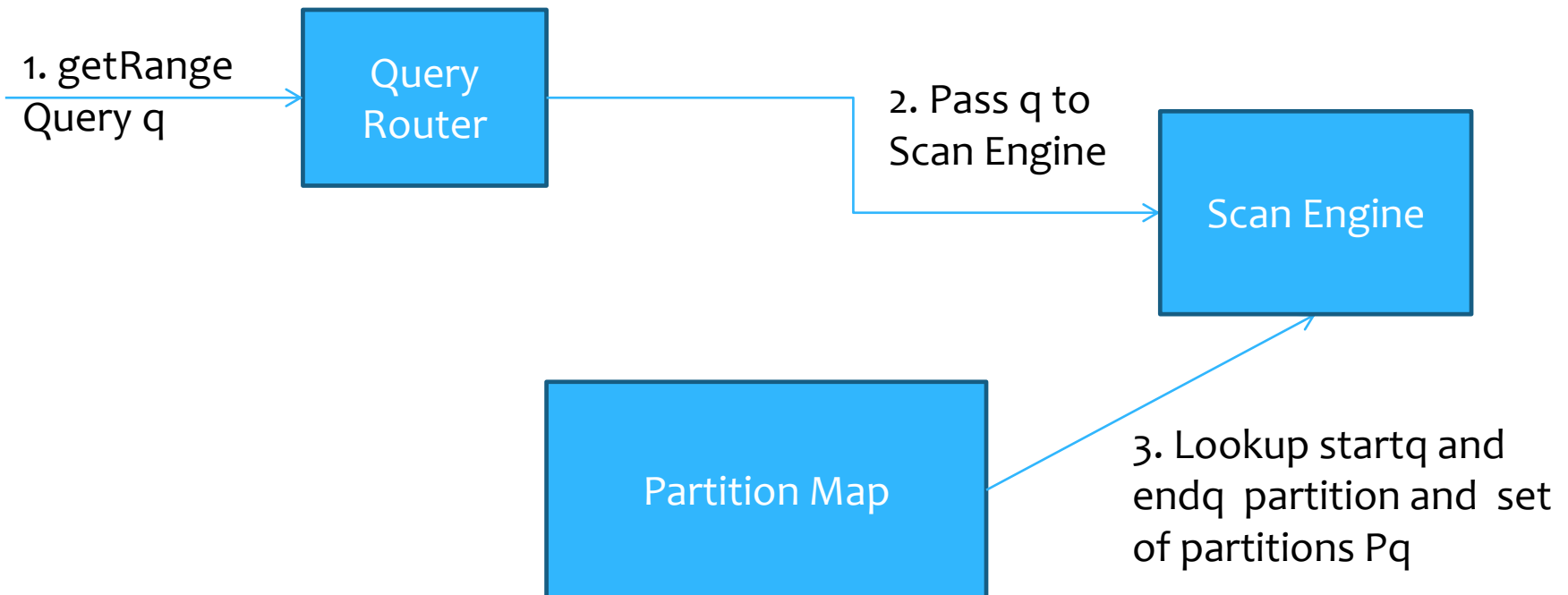
Partition  
controller



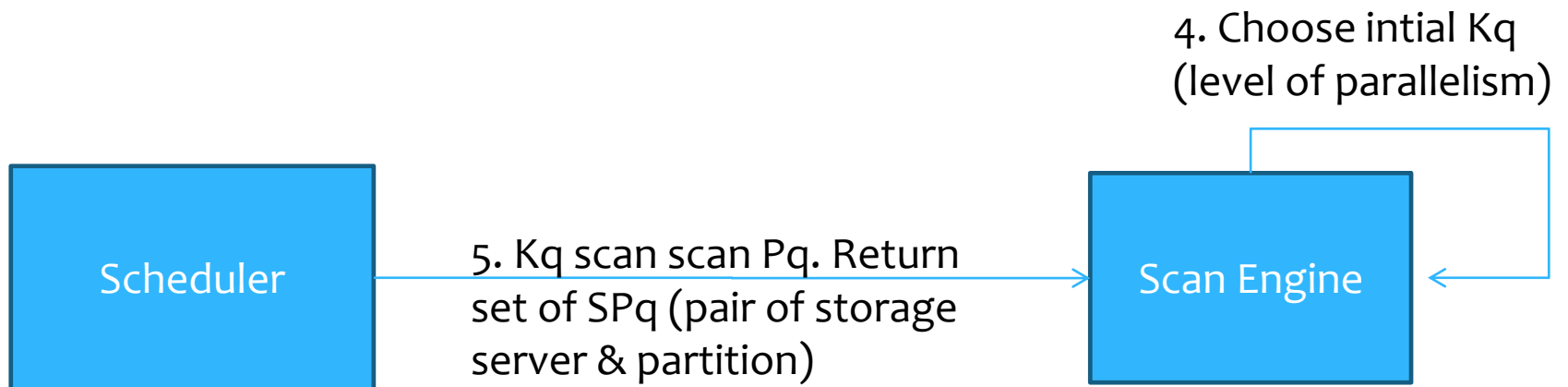
Message  
broker



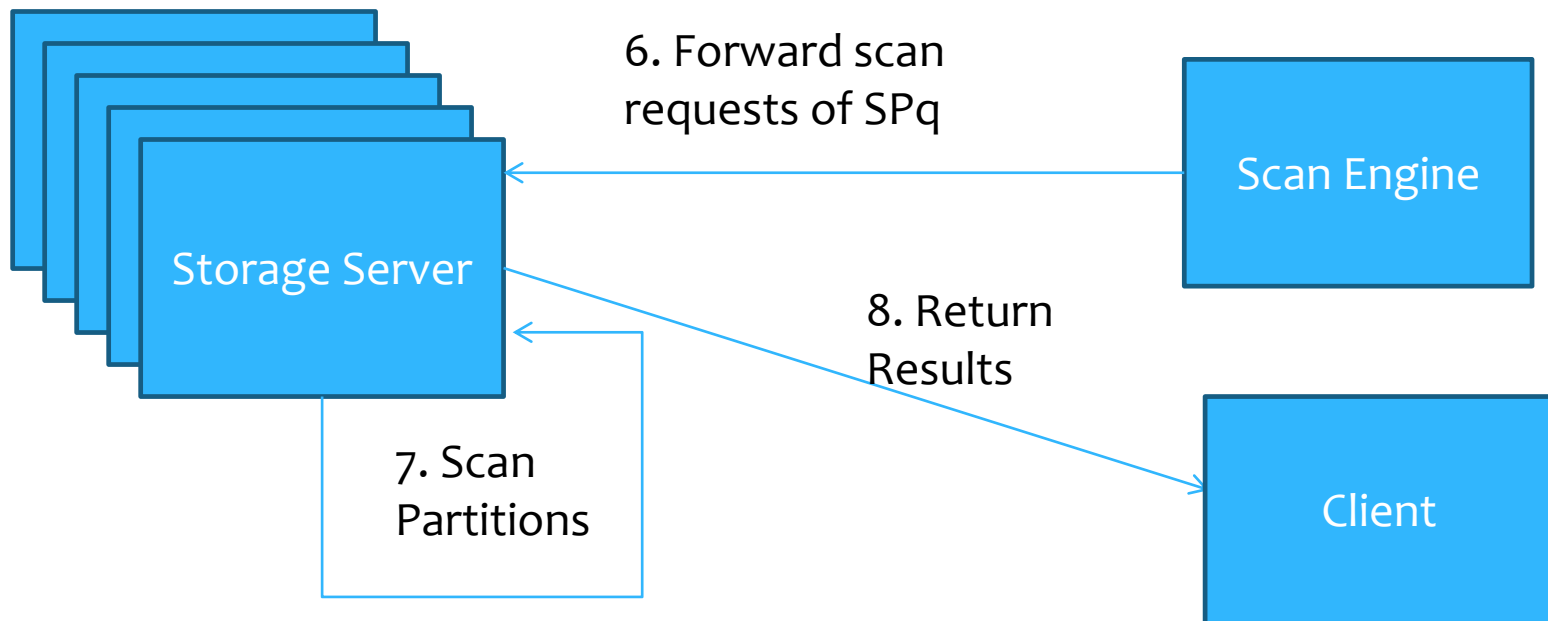
# Executing Range Queries



# Executing Range Queries (2)



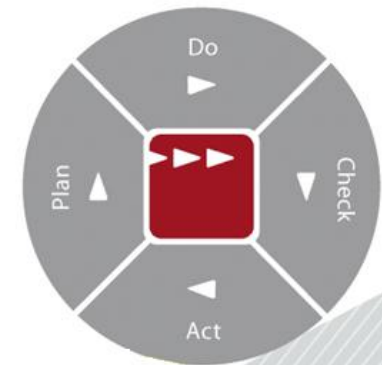
# Executing Range Queries (3)



# Agenda



- \* Engineering a new Database
  - \* Problem Description & Requirements
  - \* PNUTS: System Design & Architecture
- \* **Optimize Parallelism**
  - \* Adaptive Server Allocation Algorithm
  - \* Server Scheduling
- \* Evaluation



Level of Parallelism

# Adaptive Server Allocation



- \* Adaptively change the number of executing server
  - \* Each server **returns** records at specific **rate**
  - \* **Selectivity** of query
  - \* Client **consumption rate** (bandwidth, processing capacity & current load)



# Important Variables



- \*  $K_q$ : # of assigned servers to query  $q$
- \*  $C_q$ : client consumption of query  $q$
- \*  $T_c$ : static time interval (measuring)

# Allocation Algorithm: Initial Kq



1. Choose an initial number of servers  $K_q = 1$
2. Measure client consumption rate  $C_q(1)$  for  $T_c$  seconds
3. Repeat
  1. Set  $K_q(i+1) = K_q(i) + 1$
  2. Assign another server and begin scanning
  3. Measure client consumption rate  $C_q(i+1)$  for  $T_c$  seconds

While  $C_q(i+1) > C_q(i)$
4. Set  $K_q$  to  $K_q(i)$

# Allocation Algorithm: Adapting $K_q$



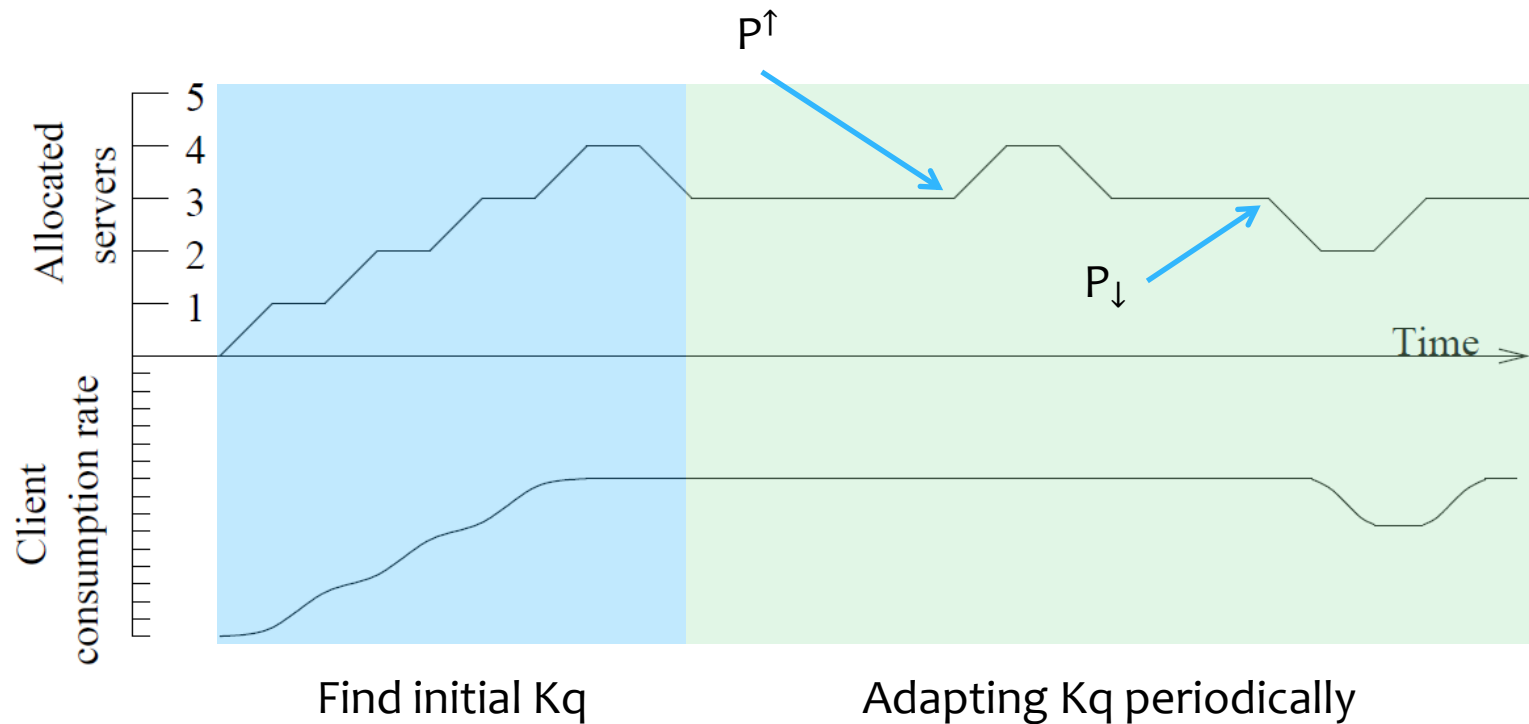
3. Every  $T_{\text{flip}}$  seconds, flip a coin
4. With probability  $P^{\uparrow}$ 
  1. Set  $K_q(i+1) = K_q(i) + 1$
  2. Allocate another server & start measure  $C_q(i+1)$  for  $T_c$
  3. If  $C_q(i+1) > C_q(i)$ , goto 4.1.
  4. Else  $K_q(i+2) = K_q(i+1) - 1$

# Allocation Algorithm: Adapting $K_q$



5. With probability  $P_{\downarrow}$ 
  1. Set  $K_q(i+1) = K_q(i) - 1$
  2. Revoke a Server & start measure  $C_q(i+1)$  for  $T_c$
  3. If  $C_q(i+1) \geq C_q(i)$ , goto 5.1.
  4. Else  $K_q(i+2) = K_q(i+1) + 1$

# Idealized Adaptive Server Allocation



# Agenda



- \* Engineering a new Database
  - \* Problem Description & Requirements
  - \* PNUTS: System Design & Architecture
- \* **Optimize Parallelism**
  - \* Adaptive Server Allocation Algorithm
  - \* **Server Scheduling**
- \* Evaluation



Level of Parallelism

# Server Scheduling



- \* Priority-Baised Round Robin
  - \* Gold vs. best-effort clients
    - \* Paid Clients will be preferred (external value)
  - \* Short/fast vs. long/slow queries
    - \* Minimum completion time

$$PBRR = \sum_{q \in Q} \frac{C + \sum_{i \in I(q)} \text{length}(i)^2}{P(q)^\alpha}$$

# Server Scheduling (2)



1. For each  $q$ , insert  $q$  into PQ with value  $\frac{C+i_q^2}{P(q)^\alpha}$
2. While servers are idle or PQ is non-empty
  1. Pop element  $q^*$  with highest priority
  2. If  $q^*$  saturated, set aside
  3. If  $q^*$  cannot use idle server, set aside
  4. Else
    1. Grant  $q^*$  one idle server
    2. Set  $i_{q^*} = 0$
    3. Insert  $q^*$  into PQ with value  $\frac{C+i_{q^*}^2}{P(q^*)^\alpha}$

# Agenda



- \* Engineering a new Database
  - \* Problem Description & Requirements
  - \* PNUTS: System Design & Architecture
- \* Optimize Parallelism
  - \* Adaptive Server Allocation Algorithm
  - \* Server Scheduling
- \* Evaluation

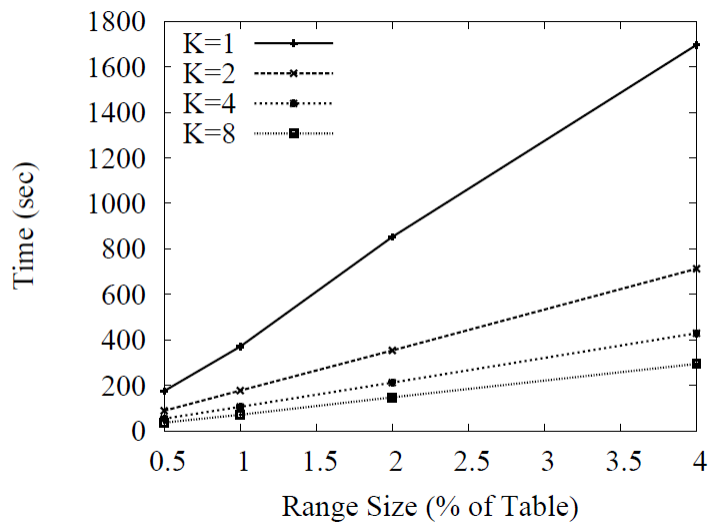


# Evaluation

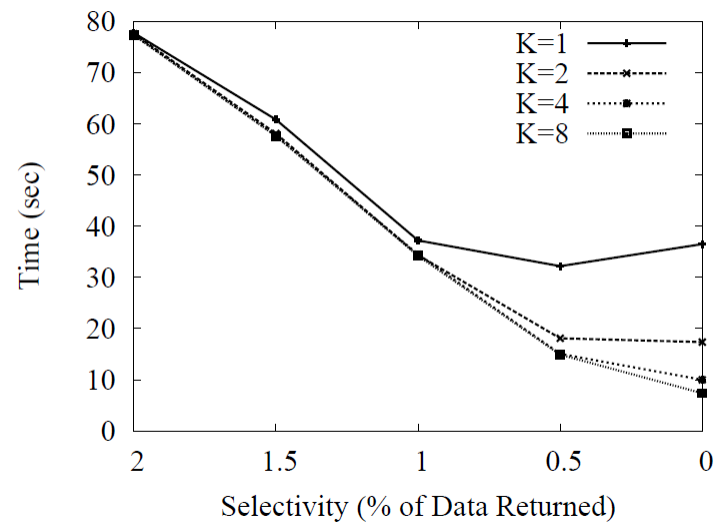


- \* Synthetic data (precise key distribution & selectivity)
- \* Metadata from Flickr (of 10 million photos)
- \* Special PNUTS client library measures client consumption rate
- \* Per-server concurrency Limit = 1

# Parallelism Impact

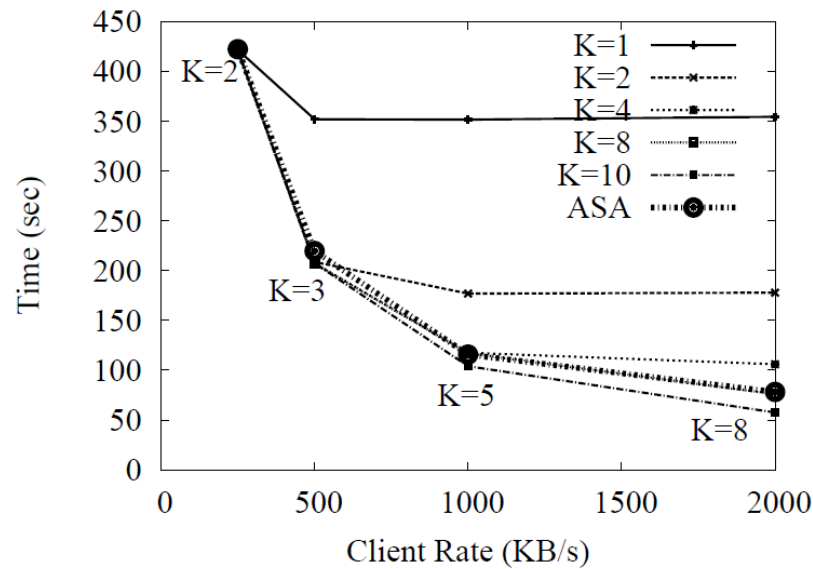


Fixed selectivity (10%) & client rate (10,000 KB/sec)



Fixed range size (1%) & client rate (250KB/sec)

# Parallelism Impact & Adaptive Server Allocation

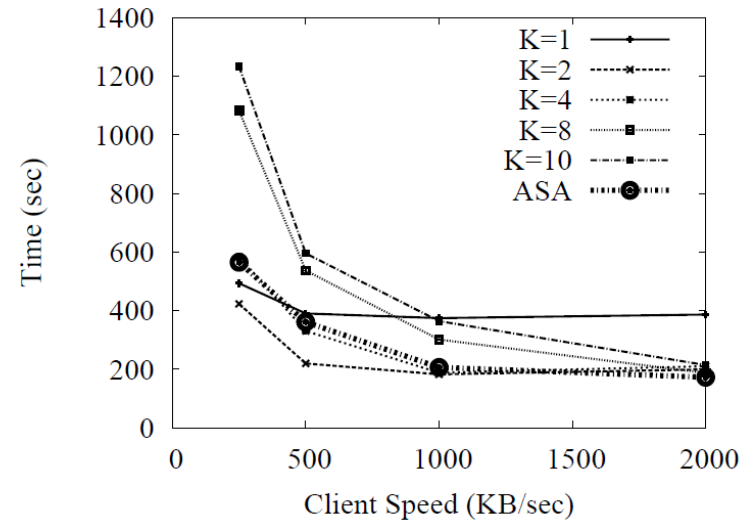
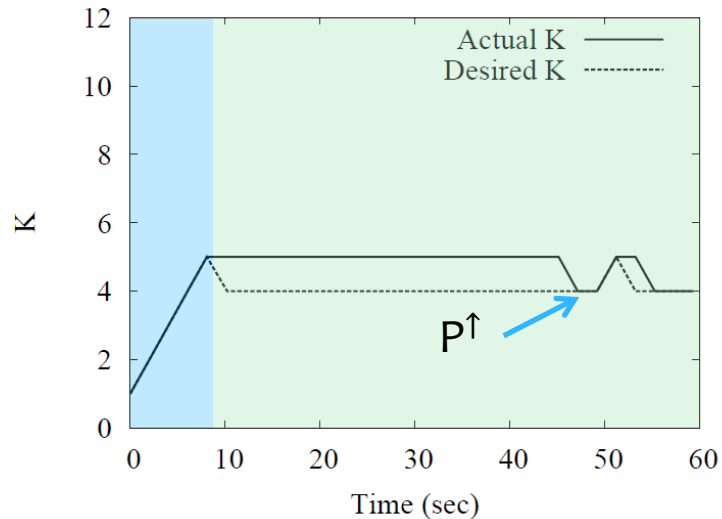


Fixed range size (1%) & selectivity (10%)

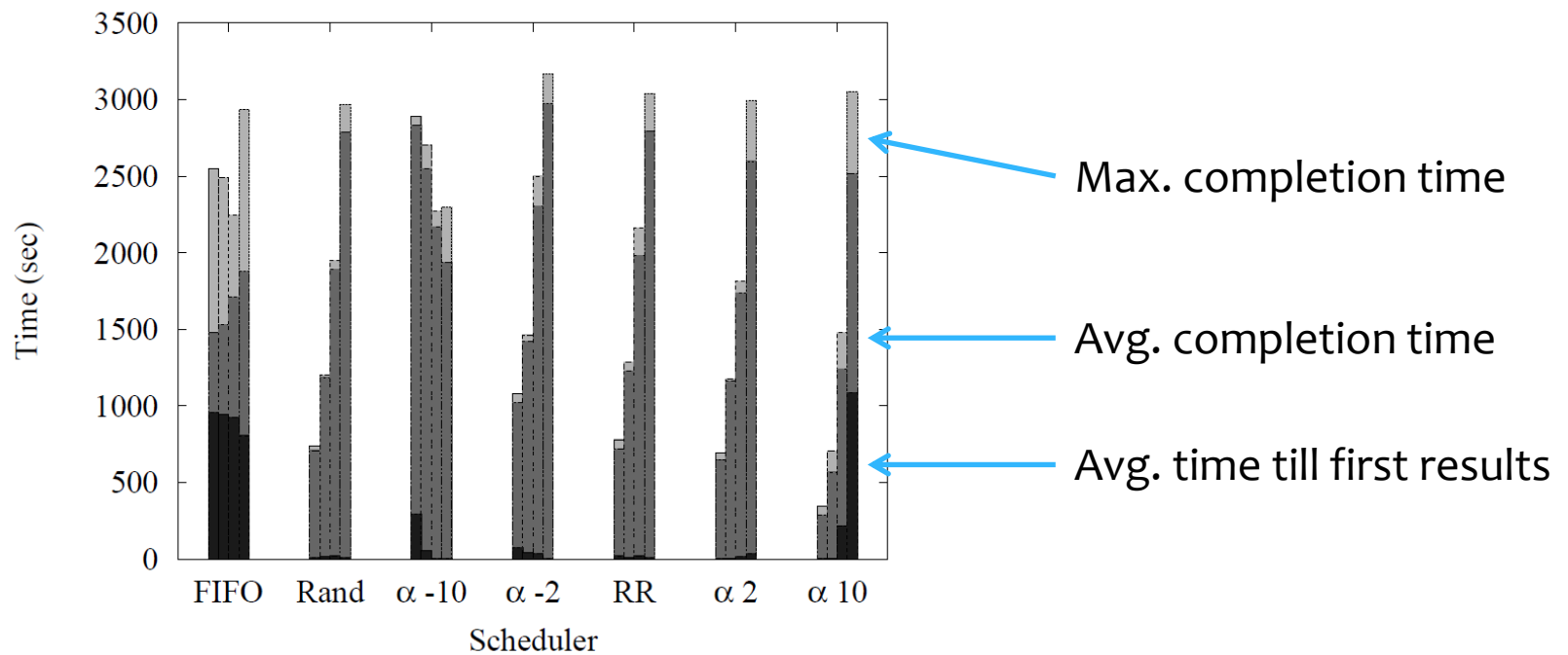
# Adaptive Server Allocation



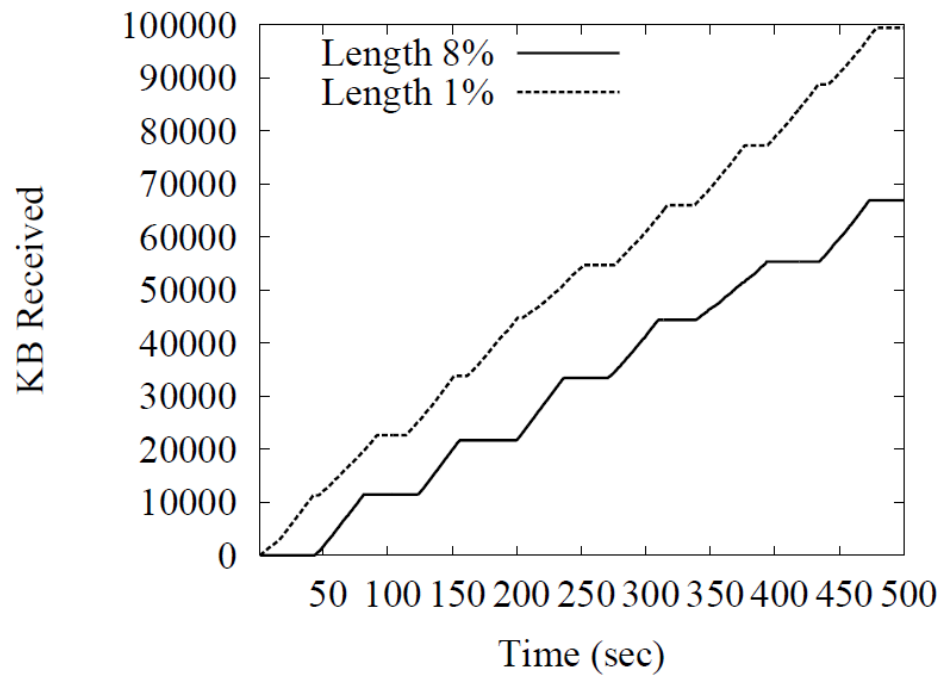
Find initial  $K_q$     Adapting  $K_q$  periodically



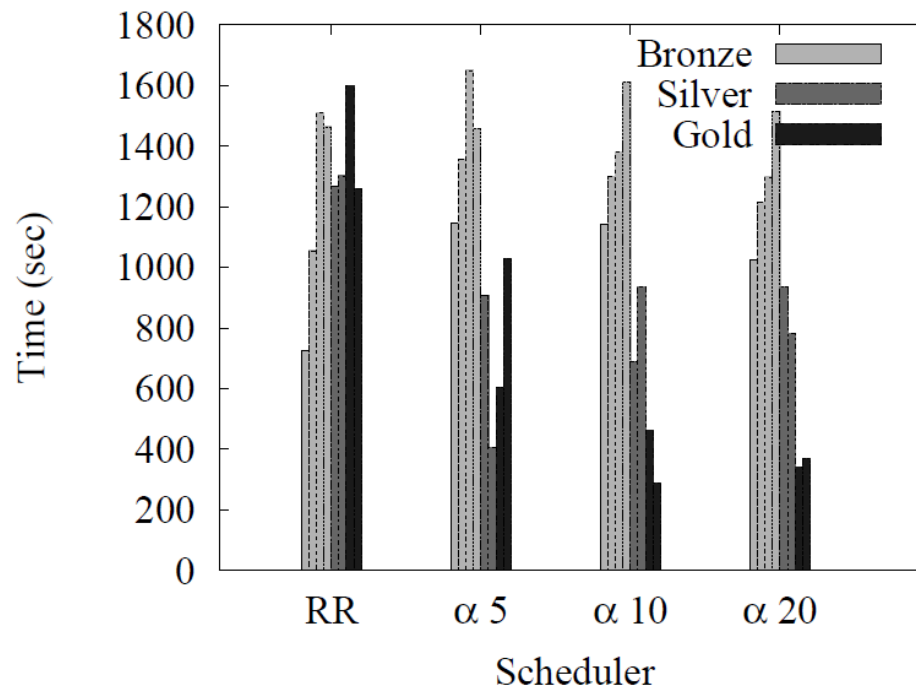
# Scheduling



# Scheduling (2)



# Scheduling (3)



# Summary



- \* **Simple** parallel database **design** for **web workload & scalability**
- \* Adopt parallelism **periodically** to client consumption rate
- \* Able to prioritize queries based on policies

Thanks for your Attention!

Any Questions?



UNIVERSITÄT  
DES  
SAARLANDES

# Backup

# Adaptive Server Allocation

