Streams, Systems and Scalability

Divesh Srivastava
AT&T Labs-Research
http://www.research.att.com/~divesh/

Map

- Streams
  - A practitioner's perspective
- Systems
- Scalability

How Much Streaming Data?

- AT&T long-distance phone network
  - ~4000 calls/sec = ~1 trillion calls in 10 years
- AT&T IP network
  - ~0.5 million flows/sec
- OC768 router
  - ~16 million packets/sec

IP Network Packet Data

```
PROTOCOL IP (Layer2) {
    uint ipversion
}
PROTOCOL IPv4(IP) {
    uint hdr_length;
    uint service_type;
    uint total_length;
    uint id;
    bool do_not_fragment;
    bool more_fragments;
    uint offset;
    uint ttl;
    uint protocol;
}
```

- Heterogeneous records
  - layer 2: ETH/HDLC
  - layer 3: IP/IPv4
  - layer 4: UDP/TCP/ICMP
  - layers 5-7: application level, e.g., HTTP, SMTP
- Analysis complicated by
  - missing packets
  - repeated packets
  - out of order packets

IP Network Application: Web Client Performance Monitoring

- Business Challenge: AT&T IP customer wanted to monitor latency observed by clients to find performance problems

- Issues
  - Use of few “active clients” is not very representative
  - Massive volumes of data (Gbit/sec links, multiple links)

- Solution: Using Gigascope data stream management system
  - Track timestamps of TCP SYN and ACK packets
  - Report latency as RTT, i.e., difference of timestamps

IP Network Application: Hidden P2P Traffic Detection

- Business Challenge: AT&T IP customer wanted to accurately monitor peer-to-peer (P2P) traffic evolution within its network

- Issues
  - Use of P2P port numbers in Netflow data is not adequate
  - P2P traffic may be “hidden” in, e.g., HTTP traffic

- Solution: Using Gigascope data stream management system
  - Search for P2P related keywords within TCP datagrams
  - Classified 3 times more traffic as P2P than Netflow

IP Network Application: Security

- Business Challenge: Alert IP customers about DDoS attacks and worms by monitoring and analyzing network data streams

- Issues
  - Massive volumes of data (Gbit/sec links, multiple links)
  - Real-time alerting (reaction time in minutes, not days)

- Solution: Using Gigascope data stream management system
  - Monitor IP traffic data streams across customer networks
  - Analyze headers + contents, identify new attack signatures

Map

- Streams
- Systems
  - Where do approximate streaming algorithms fit in?
- Scalability

Gigascope

- Gigascope is a fast, flexible data stream management system
  - High performance at speeds up to OC768 (2 x 40 Gbits/sec)
  - GSQL queries support SQL-like functionality
- Can support arbitrary data stream algorithms as UDAFs
  - GK quantile summary, count-min (CM) sketch, etc.
- Monitoring platform of choice for AT&T IP network
- Developed at AT&T Labs-Research
  - Collaboration between database and networking research

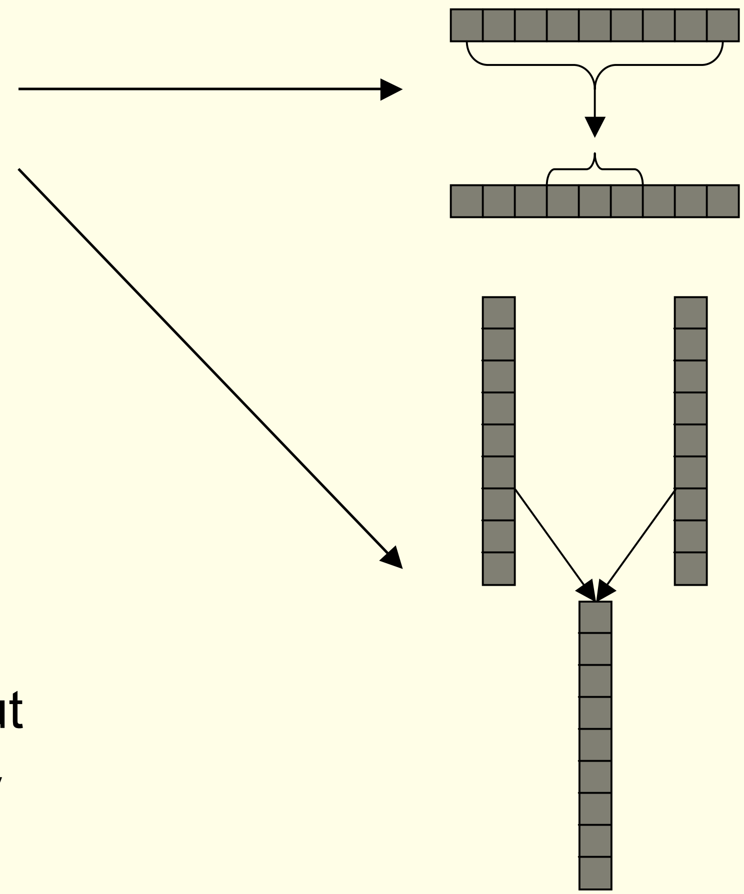
Gigascope: Data Streams

```
PROTOCOL IP (Layer2) {
    uint ipversion
}
PROTOCOL IPv4(IP) {
    uint hdr_length;
    uint service_type;
    uint total_length;
    uint id;
    bool do_not_fragment;
    bool more_fragments;
    uint offset;
    uint ttl;
    uint protocol;
}
```

- GSQL queries get raw data from low level schemas
  - defined at packet level
  - inherits from lower layer
- Current schemas include
  - layer 2: ETH/HDLC
  - layer 3: IP/IPv4
  - layer 4: UDP/TCP/ICMP
  - layers 5-7: application level, e.g., HTTP, SMTP

Gigascope: GSQL Queries

- GSQL queries support:
  - filtering, aggregation
  - merges and joins

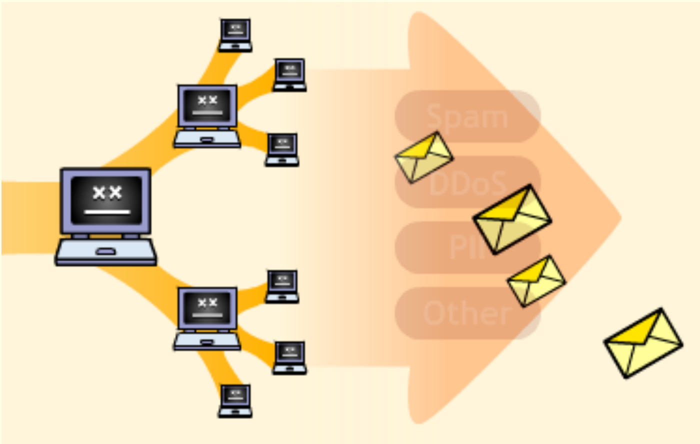


- Arbitrary code support
  - UDFs (e.g., LPM)
  - UDAFs
- GSQL query paradigm:
  - streams-in, stream-out
  - permits composability

Example: Email Bombing

- Attack characteristic: excessively many email messages
- Attack detection: monitor SMTP traffic, compare with trends
- GSQL query

```
define { query_name smtp_perhost; }
select tb, destIP, count(*), sum(len)
from TCP
where protocol = 6 and destPort = 25
group by time/60 as tb, destIP
```



Example: TCP SYN Flood

- Attack characteristic: exploits 3-way TCP handshake
- Attack detection: correlate SYN, ACK packets in TCP stream
- GSQL query

```
define { query_name toomany_syn; }
select A.tb, (A.cnt – M.cnt)
outer_join from all_syn_count A,
   matched_syn_count M
where A.tb = M.tb

define { query_name all_syn_count; }
select S.tb, count(*) as cnt
from tcp_syn S
group by S.tb
```

```
define { query_name matched_syn_count; }
select S.tb, count(*) as cnt
from tcp_syn S, tcp_ack A
where S.sourceIP = A.destIP and
   S.destIP = A.sourceIP and
   S.sourcePort = A.destPort and
   S.destPort = A.sourcePort and
   S.tb = A.tb and
   S.timestamp <= A.timestamp and
   (S.sequence_number+1) = A.ack_number
group by S.tb
```

Example: Port Scans

- Attack characteristic: probing for vulnerability
- Attack detection: track number of distinct targets probed
- GSQL query

```
define { query_name
   countdest_persource; }
select tb, sourceIP, count_distinct(
   PACK(destIP,destPort) ) as cnt
from TCP
group by time/60 as tb, sourceIP
```

```
define { query_name countdest; }
select tb, count_distinct(
   PACK(destIP,destPort) ) as cnt
from TCP
group by time/60 as tb
```

- Illustrates use of UDAFs, approximate algorithms

Gigascope: UDAF Specification

- Standard database UDAF: INIT, ITERATE, TERMINATE

- Gigascope UDAF: similar to standard database UDAF, but
  - Break TERMINATE into OUTPUT and DESTROY: enables, e.g., quantile(len, 0.9), quantile(len, 0.95), quantile(len, 0.99)

- Can support arbitrary data stream algorithms as UDAFs
  - GK quantile summary, CKMS (biased) quantile summary
  - Count-min (CM) sketch

Related DSMS Technologies

System	Data Stream Architecture	Data Model	Query Language	Query Answers	Query Plan
Aurora StreamBase	low-level	RS-in RS-out	Operators	approximate	QoS-based, load shedding
Gigascope	two level (low, high)	S-in S-out	GSQL	approximate	decomposition, distribution
Hancock	high-level	RS-in R-out	Procedural	exact, signatures	optimize for I/O, process blocks
Nile	high level	RS-in RS-out	SQL-based	approximate	incremental evaluation, multi-query
STREAM	low-level	RS-in RS-out	CQL	approximate	optimize space, static analysis
Telegraph	high-level	RS-in RS-out	SQL-based	exact	adaptive plans, multi-query

Map

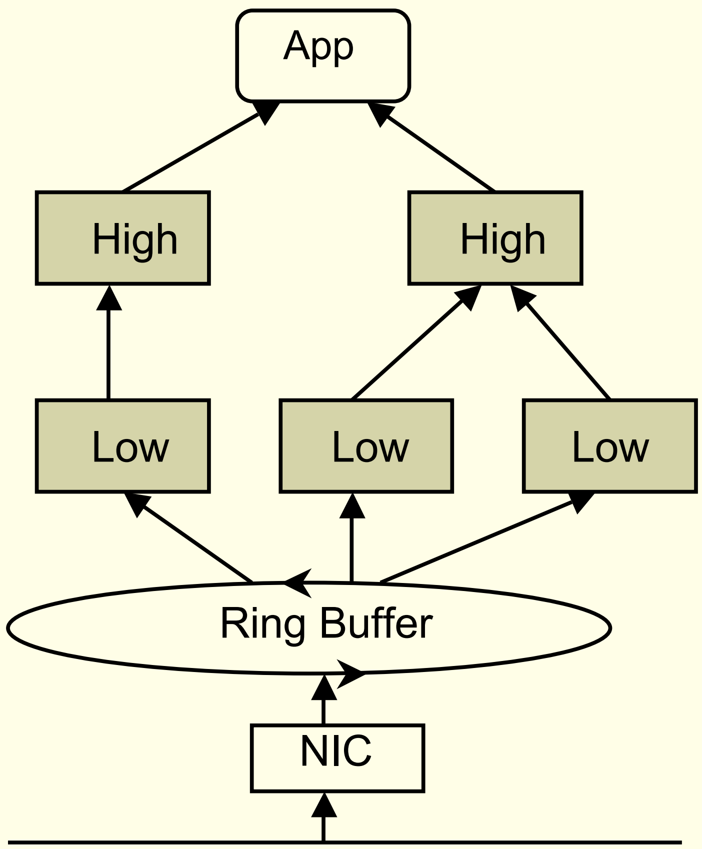
- Streams
- Systems
- Scalability
  - Opportunities for data streams research

Gigascope: Scalability

- Gigascope is a fast, flexible data stream management system
  - High performance at OC768 speeds (2 * 40 Gbit/sec)
  - Non-trivial queries at 200,000 pkts/sec using 38% of 1 CPU
- Scalability mechanisms
  - Two-level architecture: Query splitting, pre-aggregation
  - Distribution architecture: Query-aware stream splitting
  - Unblocking: Reduce data buffering
  - Sampling algorithms: Data reduction

Gigascope: Two-Level Architecture

- Low-level queries perform fast selection, aggregation
- High-level queries complete complex aggregation



Gigascope: Query Splitting

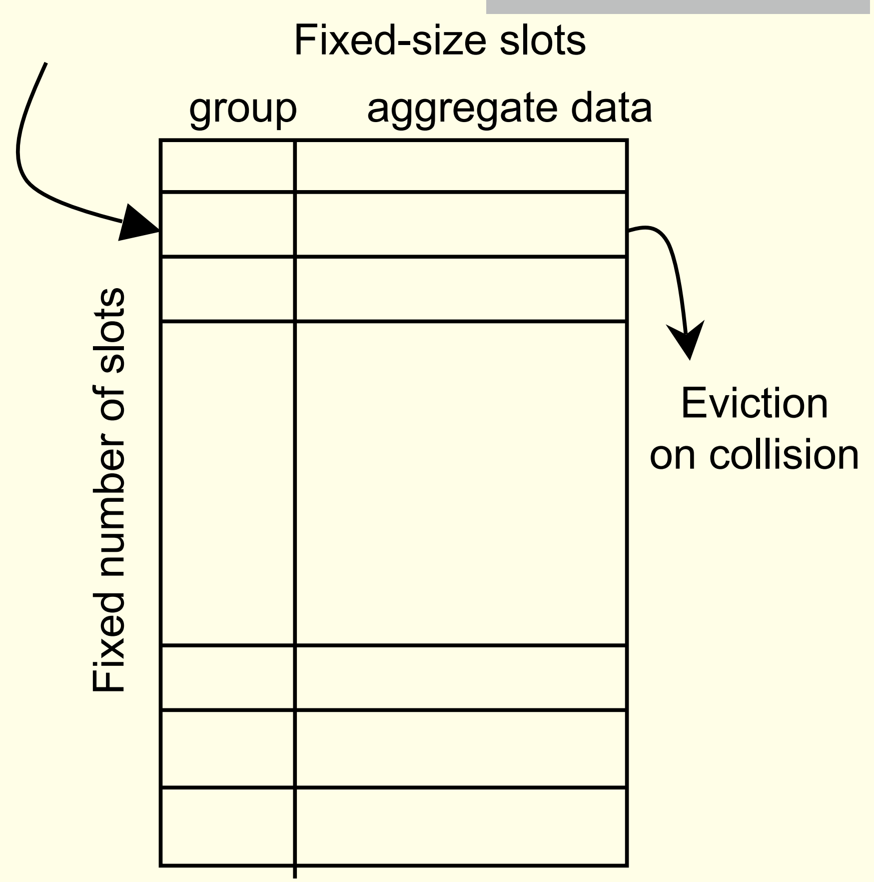
```
define { query_name smtp; }
select tb, destIP, sum(len)
from TCP
where protocol = 6 and
    destPort = 25
group by time/60 as tb, destIP
having count(*) > 1
```

```
select tb, destIP, sum(sumLen)
from SubQ
group by tb, destIP
having sum(cnt) > 1
```

```
define { query_name SubQ; }
select tb, destIP, sum(len) as
    sumLen, count(*) as cnt
from TCP
where protocol = 6 and
    destPort = 25
group by time/60 as tb, destIP
```

Gigascope: Low-Level Aggregation

- Fixed number of slots for groups, fixed size slot for each group
- Direct-mapped hashing
- Optimizations
  - Limited hash chaining reduces eviction rate
  - Slow eviction of groups when epoch changes



Gigascope: UDAF Specification

- Standard database UDAF: INIT, ITERATE, TERMINATE
- Gigascope UDAF: similar to standard database UDAF, but
  - Break TERMINATE into OUTPUT and DESTROY: enables, e.g., quantile(len, 0.9), quantile(len, 0.95), quantile(len, 0.99)
  - FLUSHME callback at the low level: used to evict a group if its aggregates become too large for fixed-size slot
- Separate UDAF code for low and high levels
  - Currently specified by UDAF creator
  - Macro language to support query splitting

Gigascope: UDAF Design Issues

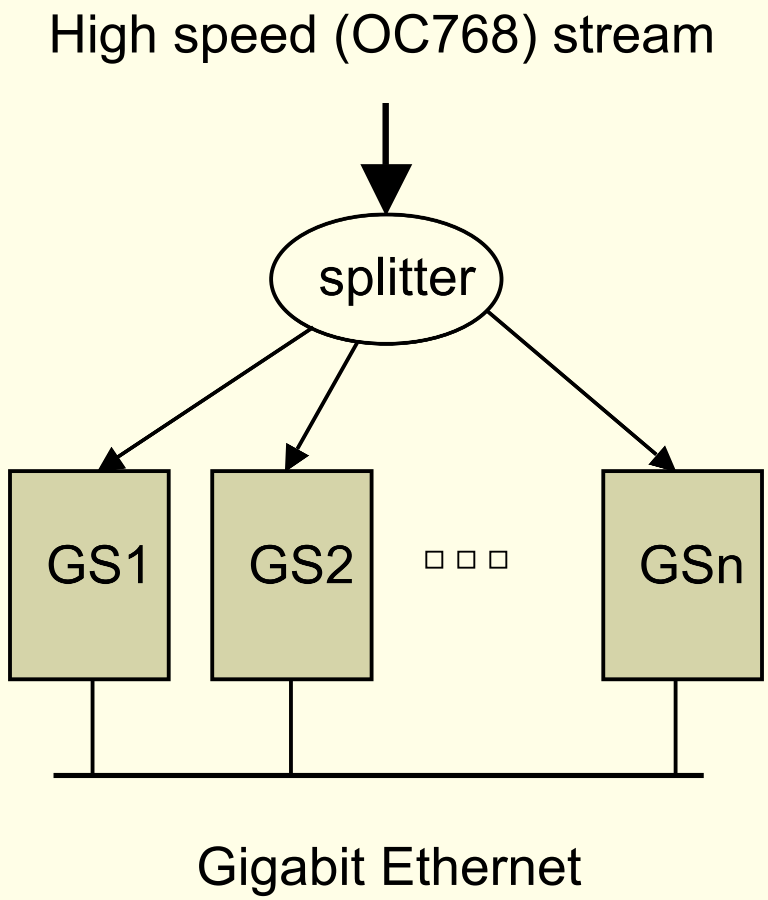
- Split processing effort between high and low level
- Processing at low-level saves processing at high-level
  - Data reduction, fewer transfers, fewer merges, etc.
- Too much processing at low-level causes packet drops
  - Quick-and-dirty filtering and aggregation
- Need to strike the right balance
  - Lightweight data structures, especially at low level
  - Avoid excessive processing at bottlenecks

Gigascope: Performance

Query	Low	High	Packets/sec
counting only	8%	0%	145,000
grouping aggregation	12.6%	0.5%	145,000
inverse distribution	25%	15.5%	142,000
UDAF	30%	43%	141,000
DDoS (join)	16.9%	3.1%	142,000
P2P (content)	10.7%	0%	139,000

Distributed Gigascope

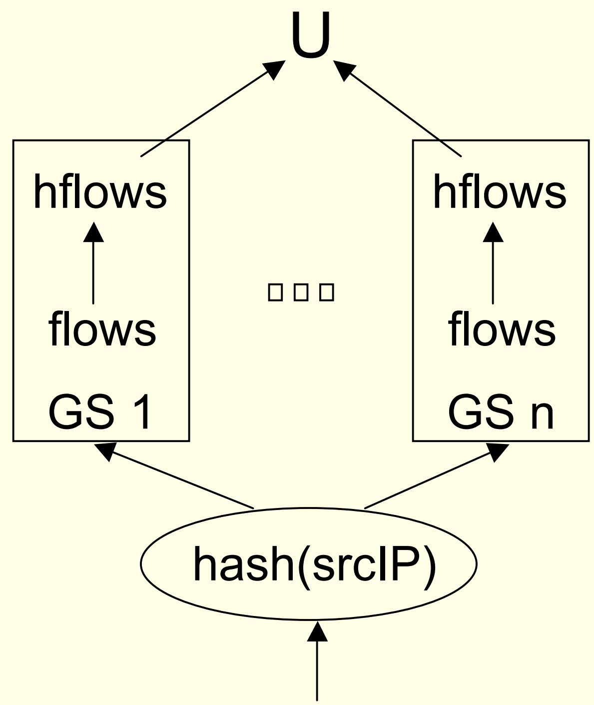
- Problem: OC768 monitoring needs more than one CPU
  - 2x40 Gb/s = 16M pkts/s
- Solution: split data stream, process query, recombine partitioned query results
- For linear scaling, splitting needs to be query-aware



Gigascope: Query-Aware Splitting

<u>define</u> { query_name flows; }
<u>select</u> tb, srcIP, destIP, count(*)
<u>from</u> TCP
<u>group by</u> time/60 as tb, srcIP, destIP

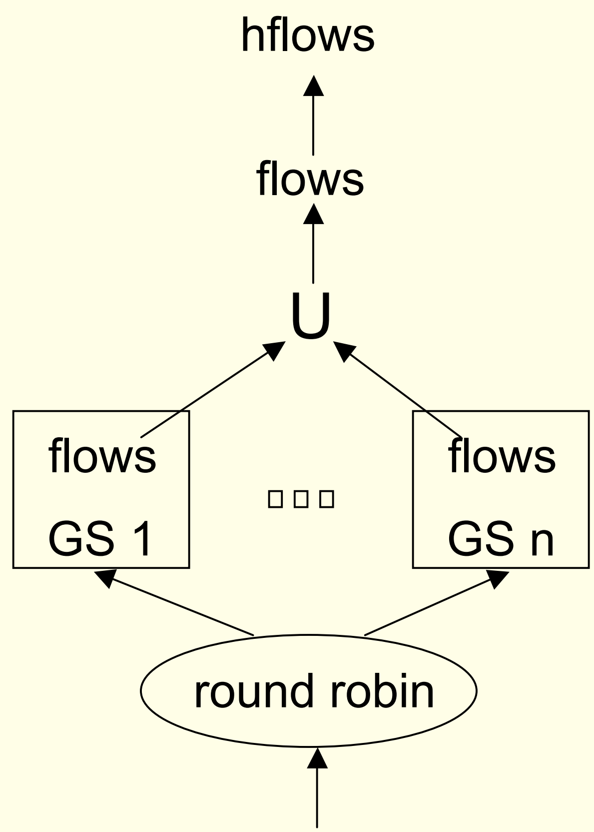
<u>define</u> { query_name hflows; }
<u>select</u> tb, srcIP, max(cnt)
<u>from</u> flows
<u>group by</u> tb, srcIP



Gigascope: Query-Unaware Splitting

define { query_name flows; }
select tb, srcIP, destIP,
    count(*)
from TCP
group by time/60 as tb, srcIP,
    destIP

define { query_name hflows; }
select tb, srcIP, max(cnt)
from flows
group by tb, srcIP



Challenges and Opportunities

- Challenges
  - Large query sets: 100s of GSQL queries, black-box UDAFs
  - Data quality: inadequate understanding of network protocols
  - Network speeds increasing: OC48 → OC192 → OC768

- Opportunities
  - Multi-query optimization: predicates, joins, UDAFs, etc.
  - Stream integrity: PAC constraints, etc.
  - Using specialized hardware: GPUs, FPGAs, etc.

Multi-Query Optimization

- Challenge
  - 100s of GSQL queries, black-box UDAFs
- Traditional MQO problem: predicates, aggregates, joins, etc.
  - Fast identification of queries relevant to a record
- Novel MQO problem: optimizable, shareable UDAFs
  - Example: GSQL queries using different sampling strategies
  - Declarative characterization (specification?) of UDAFs

Stream Integrity

- Challenge
  - Complex protocols, inadequate understanding in practice
- Queries can return inexplicable results
  - Unlike in a DBMS, cannot go back to explore the raw data
- Need to formally characterize and monitor query pre-conditions
  - Example: stream sorted on time? multiple SYN packets?
  - PAC constraints to approximately quantify violations

Using Specialized Hardware

- Challenge
  - Network speeds increasing: OC48 → OC192 → OC768
- Using commodity hardware
  - GPUs for highly parallel computations with spatial locality
- Using specialized hardware
  - FPGAs to parse TCP packet headers
  - RegEx matchers to access application-level (HTTP) fields

Conclusions

- Good news: data stream algorithms supported in Gigascope
- Better news: Gigascope and UDAFs used in practice
- Best news: scalability issues provide new opportunities!

Acknowledgements

- Colleagues
  - Graham Cormode, Ted Johnson, Flip Korn, Nick Koudas, S. Muthukrishnan, Oliver Spatscheck

- Papers and tutorials
  - Data stream query processing tutorials at VLDB'03, ICDE'05
  - Papers in SIGMOD'03, VLDB'03, SIGMOD'04, ICDE'05, SIGMOD'05, DBSec'05, VLDB'05, PODS'06