Sharing Aggregate Computation for Distributed Queries

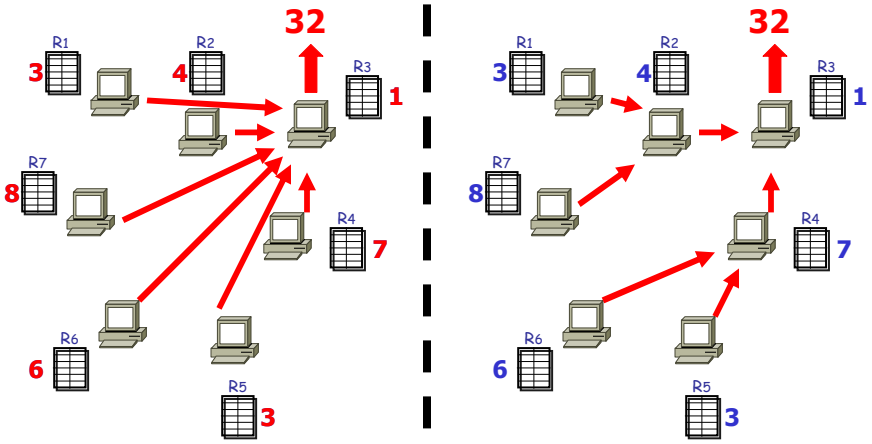
Ryan Huebsch, UC Berkeley
Minos Garofalakis, Yahoo! Research†
Joe Hellerstein, UC Berkeley
Ion Stoica, UC Berkeley

† Work done at Intel Research Berkeley

SIGMOD 6/13/07

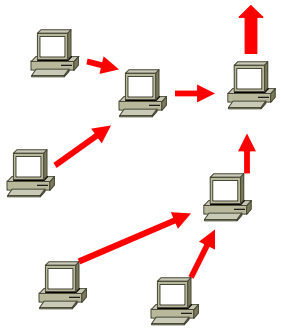
Distributed Aggregation

- SELECT count(*) FROM tableR



Cost of Aggregation

- Bottleneck is normally ***network communication***
- Query requires ***1 network message (partial state record) per node***
  - ***q*** unique queries ➔ requires ***q*** messages per node?
  - No, it can be done in fewer than ***q*** messages!



Goal of this Work

- Share network communication for multiple aggregation queries that have varying selection predicates
  - SELECT agg(a) FROM tableR WHERE <predicate>
  - Paper shows how technique applies to
    - Multiple aggregation functions
    - Queries w/ GROUP BY clauses
    - Continuous queries w/ varying windows
- Outline
  - Example and Intuition
  - Architecture
  - Techniques
  - Evaluation

Example

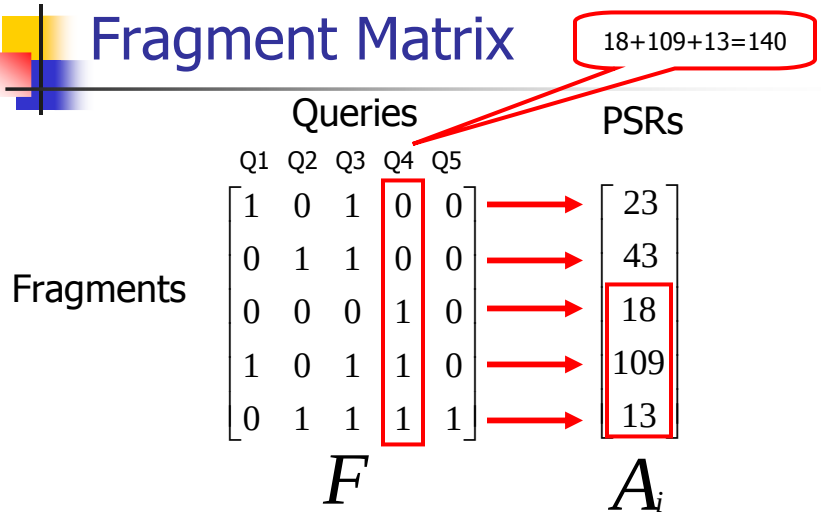
- 5 queries: SELECT count(*) FROM tableR
  - WHERE noDNS = TRUE
  - WHERE suspicious = TRUE
  - WHERE noDNS = TRUE OR suspicious = TRUE
  - WHERE onSpamList = TRUE
  - WHERE onSpamList = TRUE AND suspicious = TRUE
- Options:
  - Run each independently
  - Statically analyze predicates for containment
  - ***Dynamically analyze queries AND the data***

Fragments

- Build on Krishnamurthy's [SIGMOD 06] centralized scheme: examine which queries each tuple satisfies:
  - Some tuples satisfy queries 1 and 3 only
  - Some tuples satisfy queries 2 and 3 only
  - Some tuples satisfy query 4 only

  Etc…
- These are called ***fragments***
- The set of fragments formed by the data and queries can be represented as a matrix, F

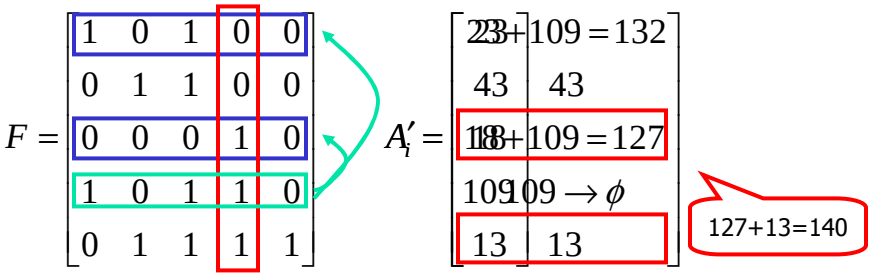
Fragment Matrix



$$A_i^{\mathrm{T}} \times F = \text{Query Answers}$$

Challenge

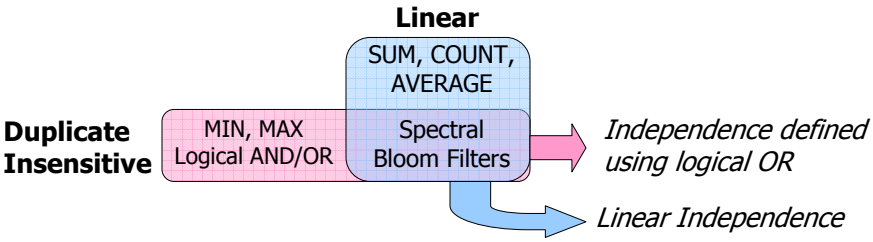
- Find a reduced A_i' which has empty entries, but still produces the correct answers



$$A_i'^{\mathrm{T}} \times F = \text{Query Answers}$$

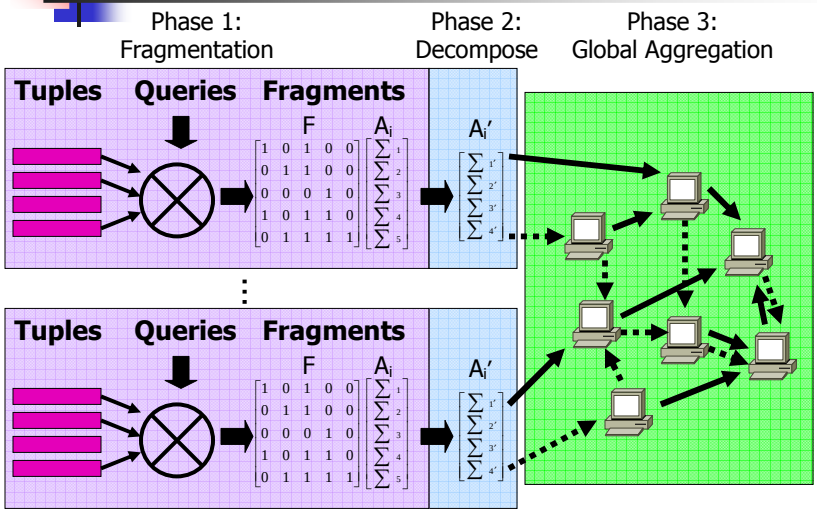
Independence

- The optimal size for A' is equal to the number of independent vectors in F
- The notion of independence varies based on the aggregation function

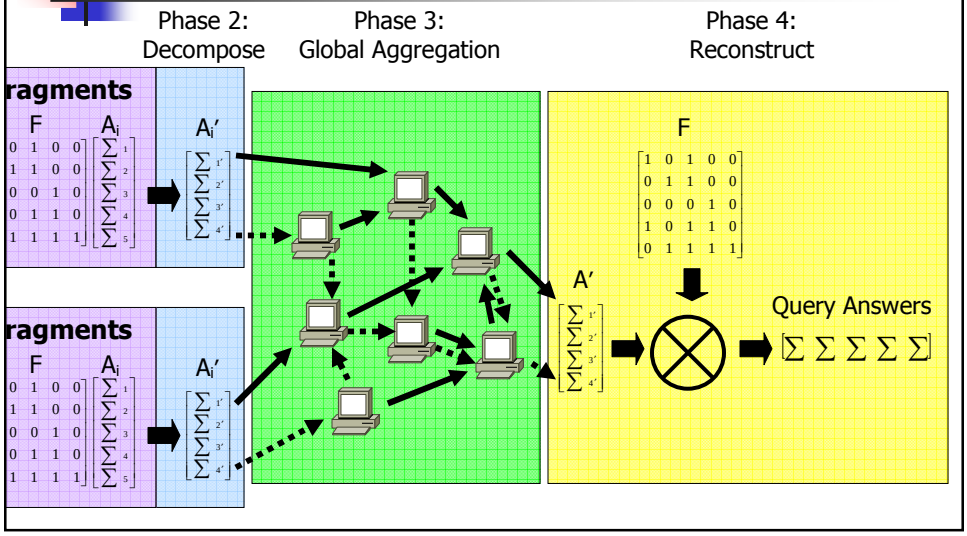


- Other aggregates (such as k-MAX/MIN) define independence using a 3-valued logic function

Architecture



Architecture



Linear Aggregates

Queries: SELECT count(*) FROM tableR WHERE <predicate>

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ & & & & 0 \\ & & & & \\ & & & & 0 \end{bmatrix} \quad A = \begin{bmatrix} 13 \\ 54 \\ 24 \\ 78 \\ 32 \end{bmatrix}$$

Entries in F′ can be any scalar number

$$A^{\mathrm{T}} \times F = Q = \begin{bmatrix} 177 & 147 & 110 & 123 & 13 \end{bmatrix}$$

13+54+24+32

compose

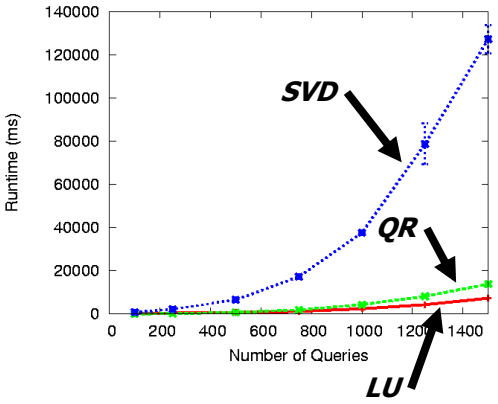
$$F' = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \end{bmatrix} \quad A' = \begin{bmatrix} 177 \\ -30 \\ 37 \\ 26 \end{bmatrix}$$

$$A'^{\mathrm{T}} \times F' = Q = \begin{bmatrix} 177 & 147 & 110 & 123 & 13 \end{bmatrix}$$

177+(-30)-37+(.5)26

Linear Aggregates

- Use ***rank revealing*** decomposition algorithms from the numerical computing literature
- LU, QR, SVD are the most common
  - Theory: All have ~O(n^3) running time, find optimal answer
  - Practice: LU is non-optimal due to rounding errors in F.P. computation, running times vary by order of magnitude



Duplicate Insensitive Aggregates

Queries: SELECT max(*) FROM tableR WHERE <predicate>

$$F = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix} \quad A = \begin{bmatrix} 25 \\ 55 \\ 79 \\ 33 \\ 14 \end{bmatrix}$$

Decompose

$$F' = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad A' = \begin{bmatrix} 55 \\ 33 \\ 79 \end{bmatrix}$$

$$A^{\mathrm{T}} \times F = Q = \begin{bmatrix} 55 & 33 & 79 & 55 & 79 \end{bmatrix}$$

max(25,55,33,14)

$$A'^{\mathrm{T}} \times F' = Q = \begin{bmatrix} 55 & 33 & 79 & 55 & 79 \end{bmatrix}$$

max(55,33)

Duplicate Insensitive Aggregates

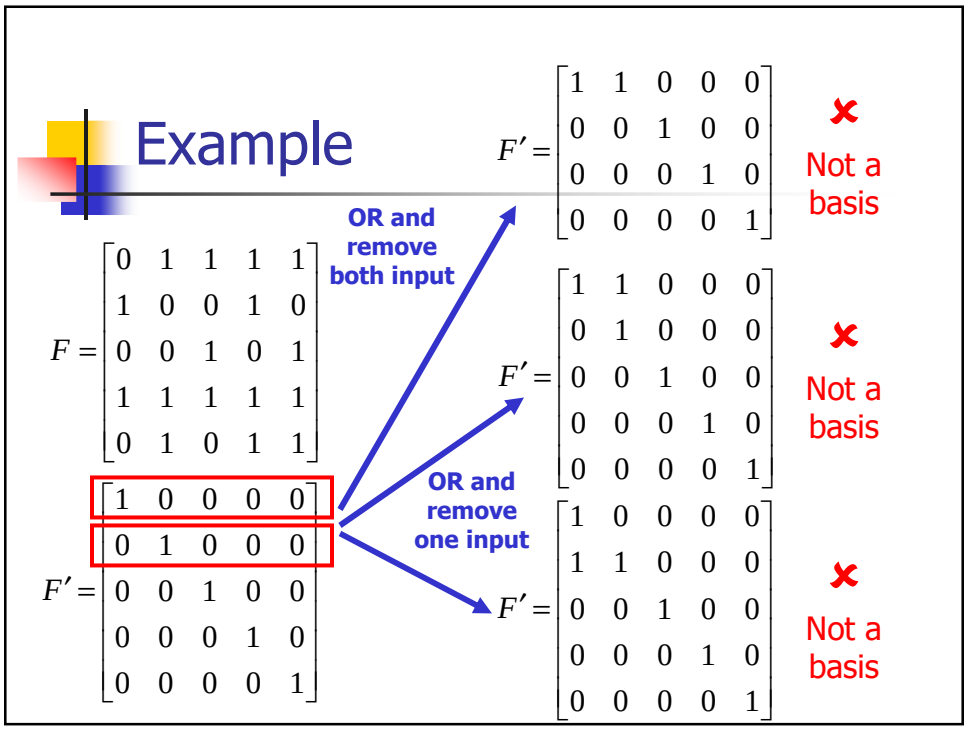
- Given F, find an F′ with fewer rows such that every vector in F can be composed by OR'ing vectors in F ➔ F′ is the Set Basis of F

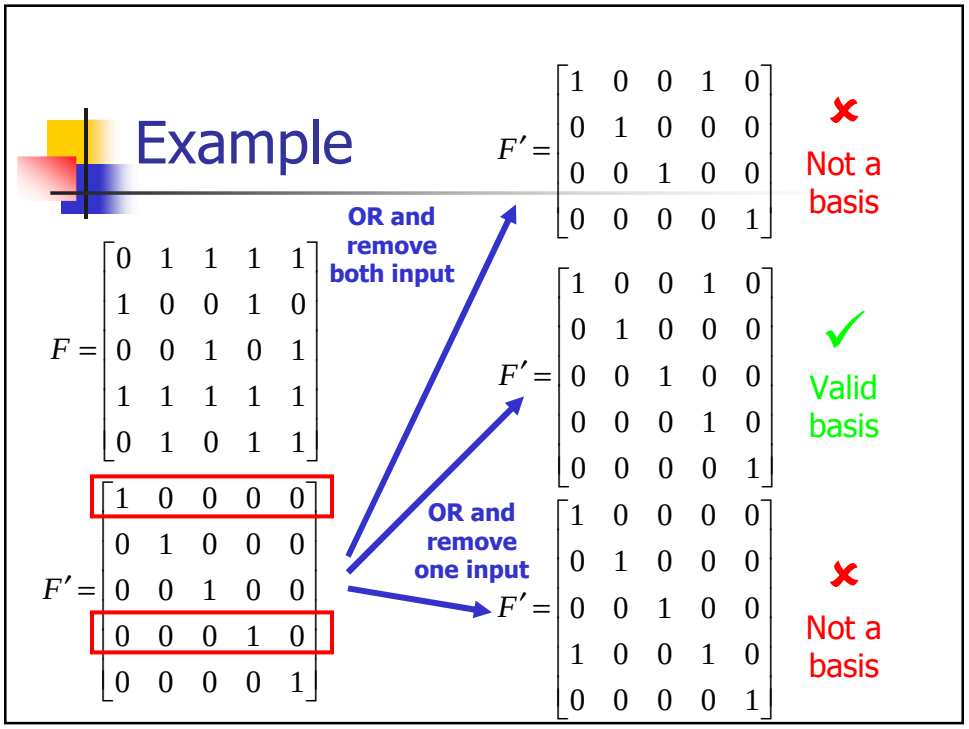
$$F = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix} \qquad F' = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

- Proven NP-Hard in 1975, No good approximations (both in theory and practice)
  - Except in very specific cases which don't apply here

Heuristic Approach

- Start with F′ as the identity matrix
  - This is equivalent to a no-sharing solution
- Apply transformations to F′
  - Simplest transformations
    - OR two vectors in F′
    - Remove a vector from F′
  - Exhaustive search takes $O(_2{}^{2^n})$
- We apply two constraints for each OR transformation
  - At least one vector must be removed
  - F′ must be always valid set-basis solution
- This significantly reduces the search space, but may eliminate the optimal answers


Example
$F = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$
$F' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
OR and remove both input
OR and remove one input
$F' = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
Not a basis
$F' = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
Not a basis
$F' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
Not a basis


Example
$F = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$
$F' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
OR and remove both input
OR and remove one input
$F' = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
Not a basis
$F' = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
Valid basis
$F' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$
Not a basis

Example

$$F = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$F' = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

OR and remove both input

$$F' = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

✓ Valid basis

Continue applying transformations exhaustively

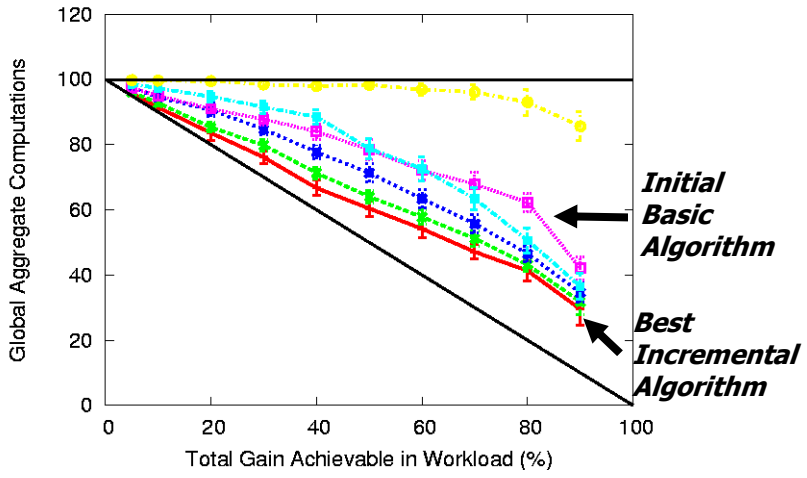
Optimizations

- The order the transformations are applied effects ***effectiveness*** and ***runtime***
  - We developed 12 strategies and evaluated them
- For 100 queries ~O(minutes), 50% optimal
- Optimization: Add each query incrementally
  - Start with 2 queries, optimize fully
  - Add another query
    - Add the identity vector for the new query to F'
    - Optimize but only consider transformations that involve new vectors
    - Repeat adding one query at a time
  - Each "mini" optimization can use any of the 12 strategies for determining order

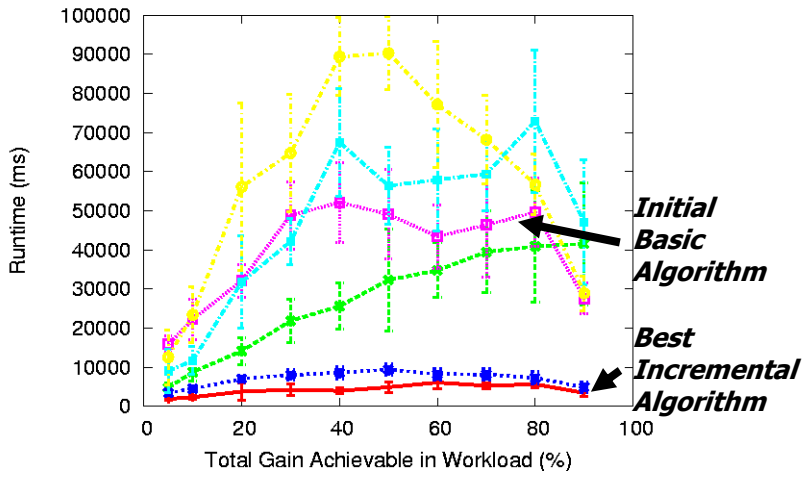
Evaluation

- Random Generator
  - Evaluate a wide range of possible workloads
  - Control the degree of possible sharing/gain
  - Know the optimal answer
- Implementation
  - Java on dual 3.06GHz Pentium 4 Xeon
- Data is averaged over 10 runs, error bars show 1 standard deviation
- Complete results in the paper

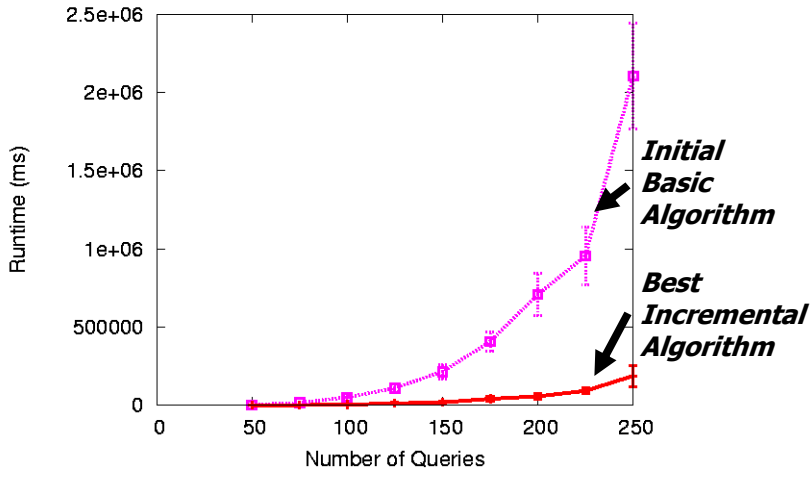
Duplicate Insensitive



Duplicate Insensitive



Duplicate Insensitive



Conclusion

- Analyzing data and queries enables sharing
- Type of aggregate function determines optimization technique
  - Linear ➔ Rank-revealing numerical algorithms
  - Duplicate Insensitive ➔ Set-basis, heuristic approach
- Very effective and modest runtimes up to
  - 500 queries with linear aggregates
  - 100 queries with duplicate insensitive aggregates