class 15

hash joins

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS165/
give me all students enrolled in cs165

```sql
select student.name from students, enrolled, courses
where courses.name = 'cs165' and enrolled.courseld = course.id and student.id = enrolled.studentId
```

CS15, Fall 2017
Stratos Idreos
The diagram illustrates a join operation. The input data is shown on the left and the join result is shown on the right. The join process combines records from two tables based on a common key. The join result includes all combinations of keys and payloads from the original tables. For example, the join result includes records like (1, d1, e1, f1, a1, b1, c1) and (2, d2, e2, f2, a2, b2, c2).
new resL[]; new resR[]; k=0
for (i=0; i<L.size; i=i++)
  for (j=0; j<R.size; j++)
    if L[i] == R[j]
      resL[k] = i
      resR[k++] = j
oracle
search in O(1)
hash join
hash join

join input 1

hash

hash table

join input 2
hash join

on which of the two inputs and when do we build the hash table
val (key), pos

N keys
val (key), pos

N keys

hash
$val (key), pos$

$N$ keys

$hash$ table

$k$ buckets
val (key), pos

h = f(val)
bucket = h mod k

N keys
k buckets

hash table
val (key), pos

hash table

h = f(val)
bucket = h \mod k

N keys

[(val1, pos1), (val7, pos7), ...]
val (key), pos

N keys

h = f(val)
bucket = h mod k

[(val1, pos1), (val7, pos7), ...]
val (key), pos

N keys

h = f(val)
bucket = h mod k

[(val1, pos1), (val7, pos7), ...]

hash table

k buckets

bucket size
bucket layout
val (key), pos

hash

h = f(val)
bucket = h mod k

[(val1, pos1), (val7, pos7), ...]

hash table

k buckets

bucket size
bucket layout

cost insert/get
val (key), pos

h = f(val)
bucket = h mod k

bucket size
bucket layout
cost insert/get

should we include pos or actual payload?

[(val1, pos1), (val7, pos7), …]
how are hash tables different than trees (leave construction aside for now)
with computation becoming less expensive compared to data movement hashing will be the winner for the foreseeable future for point queries
assumption for today:
we know how many buckets we need
(more in next class about this)
goal: join $R(val,pos)$ and $S(val,pos) = Res(posR,posS)$
goal: join $R(\text{val,pos})$ and $S(\text{val,pos}) = \text{Res}(\text{posR,posS})$

Two columns or “rows” -> same thing,
Pure columnar storage gives more flexibility typically
MonetDB moved to pure columns instead of rows of
column pairs after ~ 10 years (column pair: head, tail)
Project: pure columns
goal: join \( R(\text{val,pos}) \) and \( S(\text{val,pos}) = \text{Res}(\text{posR,posS}) \)
goal: join $R(val,pos)$ and $S(val,pos) = Res(posR,posS)$

need one page to stream $R$

Level 1
5 pages

Level 2
$R+S<<L2$
goal: join $R(val,pos)$ and $S(val,pos) = Res(posR,posS)$

need one page to stream $R$

need one page to write $Res$

Level 1
5 pages

Level 2
$R+S<<L2$
goal: join $R(\text{val}, \text{pos})$ and $S(\text{val}, \text{pos}) = Res(\text{pos}_R, \text{pos}_S)$

need one page to stream $R$

need one page to write $Res$

hash table, $S \leq L1-2$

Level 1
5 pages

Level 2
$R+S \ll L2$
goal: join $R(\text{val}, \text{pos})$ and $S(\text{val}, \text{pos}) = \text{Res}(\text{pos}_R, \text{pos}_S)$

need one page to stream $R$

need one page to write $\text{Res}$

hash table, $S \leq L_{1-2}$

Level 1
5 pages

Level 2
$R + S << L_2$

$\text{Res}$
goal: join $R(val,pos)$ and $S(val,pos) = Res(pos_{R}, pos_{S})$

- need one page to stream $R$
- need one page to write $Res$
- hash table, $S \leq L1-2$

Level 1
5 pages

Level 2
$R+S \ll L2$
goal: join $R(val,pos)$ and $S(val,pos) = Res(posR,posS)$

need one page to stream $S$

create hash table on $S$

Level 1
5 pages

Level 2
$R + S << L2$
goal: join $R(\text{val}, \text{pos})$ and $S(\text{val}, \text{pos}) = Res(\text{posR}, \text{posS})$

need one page to stream $S$

create hash table on $S$

I/O Cost in pages = $R + S + Res$
goal: join $R(val,pos)$ and $S(val,pos) = Res(posR,posS)$

**Diagram:**
- Need one page to stream $S$
- Create hash table on $S$

**Flowchart:**
- Unused

**Notation:**
- Level 1
- 5 pages

**Box:**
- Same as nested loops cost when small side fits in L1!

**Equation:**
$$I/O \text{ Cost in pages} = R + S + Res$$
what if none of the join inputs fits in L1: 
R>L1-2 & S>L1-2?

new resL; new resS; k=0;r=0;
for (i=0,i<S.size;i++,k++)
    addToHash(ht,S[i].val, S[i].pos)
    if (k==L1-2 || i==S.size-1)
        for (j=0;j<R.size;j++)
            res=probe(ht,R[i].val);
            if (res!=null)
                resR[r]=R[j].[pos]
                resS[r++]=res.pos
        empty(ht)
    k=0

hash first L1-2
values from S

scan all R
and probe

repeat until
we hash everything in S

I/O Cost in pages=S+RxS/(L1-2)+Res
what if none of the join inputs fits in L1: 
\( R > L1-2 \) & \( S > L1-2 \)?

new resL; new resS; k=0; r=0;
for (i=0;i<S.size;i++,k++)
  addToHash(ht,S[i].val, S[i].pos)
  if (k==L1-2 || i==S.size-1)
    for (j=0;j<R.size;j++)
      res=probe(ht,R[i].val);
      if (res!=null)
        resR[r]=R[j].val;
        resS[r++]=res.pos
      empty(ht)
k=0

I/O Cost in pages=S+RxS/(L1-2)+Res

This is a nested loop-like hash join

RxS factor will be an issue as data grows…
Computer science solution space:

abstractions

and at a lower level:
caching, partitioning, indirection, ..., not many more
grace hash join

join input 1

join input 2

partitioning (hash or range)
Step 1:
one pass to partition each input
grace hash join

join input 1

join input 2

partitioning (hash or range)

D1

All values in domain D1=[min1-max1]

Step 1:
one pass to partition each input
join input 1  

join input 2  

grace hash join

partitioning (hash or range)

D1

D2

All values in domain D1=[min1-max1]

All values in domain D2=[min2-max2]

Step 1:
one pass to partition each input
grace hash join

partitioning (hash or range)

Step 1: one pass to partition each input

join input 1

join input 2

All values in domain D1=[min1-max1]

All values in domain D2=[min2-max2]

All values in domain D3=[min3-max3]
grace hash join

partitioning (hash or range)

Step 1:
one pass to partition each input

All values in domain $D_1=[\text{min1}-\text{max1}]$

All values in domain $D_2=[\text{min2}-\text{max2}]$

All values in domain $D_3=[\text{min3}-\text{max3}]$

All values in domain $D_4=[\text{min4}-\text{max4}]$
grace hash join

join input 1

join input 2

partitioning (hash or range)

D1

D2

D3

D4

Step 1:
one pass to partition each input
grace hash join

Step 1:
one pass to partition each input

D1
D2
D3
D4

join input 1

join input 2

partitioning (hash or range)
grace hash join

join input 1

join input 2

partitioning (hash or range)

D1
D2
D3
D4

D1
D2

D4

Step 1:
one pass to partition each input
join input 1  
join input 2  

grace hash join

partitioning (hash or range)

Step 1:
one pass to partition each input

Step 2:
then one more pass to join each pair of partitions independently in L1
grace hash join

join input 1

join input 2

partitioning (hash or range)

Step 1:
one pass to partition each input

Step 2:
then one more pass to join each pair of partitions independently in L1
**grace hash join**

1. **Step 1:** One pass to partition each input.
2. **Step 2:** Then one more pass to join each pair of partitions independently in L1.

Diagram:
- Join input 1
- Join input 2
- Partitioning (hash or range)
- D1, D2, D3, D4

Legend:
- Blue: D1 and D3
- Red: D2 and D4
- Green: D4
join input 1

join input 2

grace hash join

partitioning (hash or range)

D1
D2
D3
D4

nothing to do here

Step 1:
one pass to partition each input

Step 2:
then one more pass to join each pair of partitions independently in L1
grace hash join

join input 1

join input 2

partitioning (hash or range)

D1
D2
D3
D4

nothing to do here

Step 1:
one pass to partition each input

Step 2:
then one more pass to join each pair of partitions independently in L1
I/O Cost?

Step 1:
- one pass to partition each input

Step 2:
- then one more pass to join each pair of partitions independently in L1

nothing to do here
I/O Cost?

I/O cost in pages = 3(R+S) + res

So now this scales with data size

**Step 1:**
- one pass to partition each input

**Step 2:**
- then one more pass to join each pair of partitions independently in L1

D1

D2

D3

D4

nothing to do here

R+S for first pass
R+S to write partitions
R+S for second pass result
1. read input into stream buffer, hash and write to respective partition buffer
2. when input buffer is consumed, bring the next one
3. when a partition buffer is full, write to L2
we can partition into L1-1 pieces in one pass
as long as at least one of the pieces $\leq L_{1-2}$
join input 1

join input 2

both left and right side >L1-2
apply recursively
if a partition
does not fit in memory
for every
L.key = R.key pair
return [L.pos,R.pos]

data/results
one column-at-a-time

1. quick analysis: when can we do grace join in 2 passes (per input) max?

2. redesign grace join to utilize multi-cores

3. keep all cores at 100% all the time
   (minimize L3 cache misses and no TLB misses)
when can we do grace join in exactly 2 passes?

1) after first pass, and for each partition, at least one side should fit in L1-2

2) simplify problem by considering one side only:
   if all partitions we create from one side fit in L1-2 we are ok

3) the maximum number of partitions we can create in one pass is L1-1

4) so if R/L1-1<=L1-2 -> R<=(L1-1)(L1-2)
   we will not need to repartition any pieces= we can join in two passes
how to partition in parallel?

join

**partition** each input

**for each** pair of partitions

**create** hash table

**probe** hash table

how to create HT in parallel?

how to probe HT in parallel?
1. compute in parallel
2. control access to output buffers (latches)
(or split data/memory pieces=partitions/cores)
1. compute in parallel
2. control access to output buffers (latches)
(or split data/memory pieces=partitions/cores)

We could load more input pages and assign them in different cores?
We could load more input pages and assign them in different cores?

How many conflicts are we going to have? How bad is it going to be?
stream input

compute hash values & buckets in parallel

Core1
Core2
Core3
Core4

hash

p1 hash table

unused

p1 inner
p1 outer
compute hash values & buckets in parallel

Core1  Core2  Core3  Core4

stream
input

hash

p1 hash table

result

p1 inner

p1 outer

p1 result

control
cache conscious grace join

- CPU
- L1
- L2
- L3
- memory
- disk

partition R such as each hash table fits in L3 (L3-2) as much as K=L3-1 partitions in one pass

every time a buffer/cache line is full spill to memory we may be writing to K memory areas

if K>TLB then we incur TLB misses
radix join

recursively partition with maximum partitions in each pass $\leq$ TLB

may do more passes but sequential access for reads (CPU happy) and random access only for writes (but $< \text{TLB}$)

$= \text{no TLB misses}$
what to do in M4?
nested loops and multi-core grace hash joins

be able to demonstrate that your joins are: cache conscious and utilize multi-core nicely
Browse: **Join Processing in Databases with Large Main Memories**
L. Shapiro
ACM Transactions on Database Systems. 11(3), 1986

Read: **Cache Conscious Algorithms for Relational Query Processing**
A. Shatdal, C. Kant and J. Naughton
Very Large Databases Conference, 1984

Read: **Database Architecture Optimized for the new Bottleneck: Memory Access**
P. Boncz, S. Manegold and M. Kersten
Very Large Databases Conference, 1999

Browse: **Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-Core CPUs**
Changkyu Kim, et all.
International Conference on Very Large Databases, 2009
hash joins

DATA SYSTEMS

prof. Stratos Idreos