data systems 101

prof. Stratos Idreos

HTTP://DASLAB.SEAS.HARVARD.EDU/CLASSES/CS265/
2 classes per week - OH/Labs every day
1 presentation/discussion lead - 2 reviews each week
research (or systems) project + midway check-in

~15-20 hours per week
systems project

individual project
c/c++

research project

groups of three
analysis

only open to cs165 students (unless proven otherwise)
noSQL key-value stores are everywhere…

project= basic state-of-the-art design
Give me a design given read/write ratio.
How new hardware X would impact performance?
What hardware is needed to achieve throughput X?
big data V’s
(it is not about size only)

volume  velocity  variety  veracity

actually none of that is really new…

new:
our ability to gather and store machine generated data
broad understanding that we cannot just manually get value out of data
a data system **stores** data and **provides access** to data

& makes knowledge generation easy
a data system **stores** data  
and **provides access** to data  

& makes knowledge generation easy
a data system **stores** data and **provides access** to data & makes knowledge generation easy
declarative interface
ask “what” you want

the system decides “how” to best store and access data

data system
applications

database kernel

algorithms/operators

data

data

sql

cpu

memory

disk

CS265, Spring 2018
Stratos Idreos
As apps become more complex and need to be more scalable, the goal is to understand why, how, and what is next in the realm of SQL and noSQL databases.

- SQL: complex, legacy, tuning, expensive...
- noSQL: simple, clean, just enough...

newSQL

---

goal: understand why, how and what is next
As apps become more complex and need to be more scalable, the goal is to understand why, how, and what is next.

**SQL**
- Complex
- Legacy
- Tuning
- Expensive

**noSQL**
- Simple
- Clean
- Just enough

Gartner: DBMS Market = ~36Billion

noSQL/Hadoop = ~700 Million

SQL = ~rest

groupbox

newSQL
more data

more applications

continuous need for new systems

more h/w
soon everyone will need to be a “data scientist”

hmm, my data is too big :(

how far away are we from a future where a data system sits in the critical path of everything we do?

new applications/requirements
data exploration

not always sure what we are looking for (until we find it)
years
[IBMbigdata]

data* skills

years
[StratosGuess]

data system design, set-up, tune, use
system where db runs

- cpu - cpu - cpu - cpu
- cpu registers
- caches
- memory
- disk - disk - disk - disk

+ flash
+ non volatile memory

memory hierarchy
Jim Gray, IBM, Tandem, DEC, Microsoft
ACM Turing award
ACM SIGMOD Edgar F. Codd Innovations award

1.5 hours
New York

10 min
this building

1 min
this room

~0
my head

10x
on board cache

2x
on chip cache

100x
memory

100Kx
disk

10 min
this building

10 min
this room

2 years
Pluto

100x
memory

2x
on board cache

10x
on chip cache

~0
my head
There are different types of memory: CPU registers, on-chip cache, on-board cache, memory (SRAM, DRAM), and disk.

- **CPU registers** are the fastest but also the most costly.
- **On-chip cache** is faster than memory but less costly than registers.
- **On-board cache** is slower than on-chip cache but faster than memory.
- **Memory** (SRAM, DRAM) is the slowest but also the cheapest.

**Cache miss** occurs when the CPU is looking for something that is not in the cache.

**Memory miss** occurs when the CPU is looking for something that is not in memory.

The **memory wall** refers to the performance gap between the CPU and memory, which is often illustrated by a graph showing speed vs. time, with CPU and memory as axes.
DON’T MISS!

and touch/access only what you need

design of storage/access methods/algorithms should minimize:

- data misses
  + instruction misses
random access & page-based access

need to only read $x$… but have to read all of page 1

data value $x$

page1  page2  page3  …

data move

CPU
registers
on chip cache
on board cache
memory
disk
query $x < 5$

(page size = 120 bytes)

memory level $N$

memory level $N-1$

5 10 6 4 12 2 8 9 7 6 7 11 3 9 6

page size: 5x8 bytes
query $x < 5$

scan

memory level N

(size=120 bytes)

memory level N-1

page size: 5x8 bytes
query $x < 5$

Scan:

(memory level N)

```
5 10 6 4 12
```

(size = 120 bytes)

(memory level N-1)

```
5 10 6 4 12
2 8 9 7 6
7 11 3 9 6
```

Page size: 5x8 bytes
query $x < 5$

memory level $N$

(size = 120 bytes)

scan

5 10 6 4 12

4

memory level $N-1$

page size: 5x8 bytes

5 10 6 4 12 2 8 9 7 6 7 11 3 9 6 ...
query $x < 5$

(size=120 bytes)

memory level N

(scan) 5 10 6 4 12

40 bytes

memory level N-1

(scan) 2 8 9 7 6

page size: 5x8 bytes
query \( x < 5 \)

(memory level N)

(size = 120 bytes)

scan

5 10 6 4 12

scan

2 8 9 7 6

4 2

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...
query $x < 5$

Scan

memory level $N$

(size=120 bytes)

5 10 6 4 12

Scan

memory level $N-1$

5 10 6 4 12

2 8 9 7 6

4 2

7 11 3 9 6

page size: 5x8 bytes
query $x < 5$

(memory level $N$)

(size=120 bytes)

memory level $N-1$

page size: 5x8 bytes
query $x < 5$

scan

(memory level N)

(size = 120 bytes)

(page size: 5x8 bytes)

(memory level N-1)
query $x < 5$

(size=120 bytes)
memory level N

scan

7 11 3 9 6 2 8 9 7 6 4 2 3

memory level N-1

5 10 6 4 12 2 8 9 7 6 7 11 3 9 6 ...

page size: 5x8 bytes

80 bytes
query $x < 5$

scan

memory level N

(size=120 bytes)

memory level N-1

page size: 5x8 bytes
an oracle gives us the positions

query $x < 5$

(size = 120 bytes)

memory level $N$

memory level $N-1$

$\begin{array}{cccccc}
5 & 10 & 6 & 4 & 12 \\
2 & 8 & 9 & 7 & 6 \\
7 & 11 & 3 & 9 & 6 \\
\end{array}$

page size: 5x8 bytes
an oracle gives us the positions

query $x < 5$

memory level $N$

\[
\text{oracle} \rightarrow \begin{bmatrix}
5 & 10 & 6 & 4 & 12 \\
2 & 8 & 9 & 7 & 6
\end{bmatrix}
\]

(size=120 bytes)

memory level $N-1$

\[
\begin{bmatrix}
5 & 10 & 6 & 4 & 12 \\
2 & 8 & 9 & 7 & 6 \\
7 & 11 & 3 & 9 & 6
\end{bmatrix}
\]

page size: 5x8 bytes
an oracle gives us the positions

query \( x < 5 \)

oracle

(size=120 bytes)
memory level N

5 10 6 4 12

4

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

page size: 5x8 bytes
an oracle gives us the positions

query \( x < 5 \)

\( \text{oracle} \)

(size=120 bytes)
memory level N

5 10 6 4 12

4

memory level N-1

5 10 6 4 12
2 8 9 7 6
7 11 3 9 6
...

page size: 5x8 bytes
an oracle gives us the positions

query \( x < 5 \)

**(size=120 bytes)**

memory level \( N \)

5 10 6 4 12

oracles

2 8 9 7 6

4

memory level \( N-1 \)

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

... page size: 5x8 bytes

40 bytes
An oracle gives us the positions

**query** \( x < 5 \)

Memory level \( N \)

```
| 5 | 10 | 6 | 4 | 12 |
```

(size = 120 bytes)

Memory level \( N - 1 \)

```
| 5 | 10 | 6 | 4 | 12 |
| 2 | 8 | 9 | 7 | 6 |
| 7 | 11 | 3 | 9 | 6 |
```

Page size: 5x8 bytes

Oracle:

```
| 5 | 10 | 6 | 4 | 12 |
```

Oracle:

```
| 2 | 8 | 9 | 7 | 6 |
| 4 | 2 |
```

 Oracle gives additional positions:

```
| 7 | 11 | 3 | 9 | 6 |
```

Oracle gives additional positions:
an oracle gives us the positions

query $x < 5$

(size=120 bytes)

memory level N

oracle

5 10 6 4 12

oracle

2 8 9 7 6

4 2

page size: 5x8 bytes

memory level N-1

5 10 6 4 12

2 8 9 7 6

7 11 3 9 6

...
an oracle gives us the positions

\[
\text{query } x < 5
\]

(size=120 bytes)

memory level N

\[
\begin{align*}
5 & \quad 10 & \quad 6 & \quad 4 & \quad 12 \\
2 & \quad 8 & \quad 9 & \quad 7 & \quad 6 \\
7 & \quad 11 & \quad 3 & \quad 9 & \quad 6
\end{align*}
\]

memory level N-1

\[
\begin{align*}
5 & \quad 10 & \quad 6 & \quad 4 & \quad 12 \\
2 & \quad 8 & \quad 9 & \quad 7 & \quad 6 \\
7 & \quad 11 & \quad 3 & \quad 9 & \quad 6
\end{align*}
\]

page size: 5x8 bytes
an oracle gives us the positions

query $x < 5$

memory level N

```
7 1 1 3 9 6
2 8 9 7 6
4 2
```

(size=120 bytes)

memory level N-1

```
5 1 0 6 4 12
2 8 9 7 6
7 1 1 3 9 6 ...
```

page size: 5x8 bytes
an oracle gives us the positions

\[ \text{query } x < 5 \]

oracle

(size=120 bytes)

memory level N

memory level N-1

page size: 5x8 bytes
an oracle gives us the positions

query \( x < 5 \)

(memory level N)

| Oracle | 7 11 3 9 6 | 2 8 9 7 6 | 4 2 3 |

(memory level N-1)

| Page | 5 10 6 4 12 | 2 8 9 7 6 | 7 11 3 9 6 | ... |

page size: 5x8 bytes
when does it make sense to have an oracle
how can we minimize the cost

e.g., query $x < 5$
it all starts with how we store data

**every bit matters**
**sequential access:**
read one block; consume it completely; discard it; read next

**in parallel/prefetching**

what is next?

1 2 3 4

hardware can better predict/buffer sequential pages to be read
e.g., 2MB buffers in modern DRAM
amortize cost of moving disk arms
random access:
read one block; consume it partially; discard it;
might have to read it again in future; read “random” next;
C/C++

no libraries unless we explicitly allow it
we expect you build everything from scratch
so you can control storage and access 100%
MAIN-MEMORY OPTIMIZED DATA SYSTEMS
a “simple” example

assume an array of $N$ integers:
find all positions where $value > x$

qualifying positions

exists in all systems: sql, nosql, newsql

select operator
assume an array of \( N \) integers: find all positions where \( \text{value} > x \)

```java
res = new array[data.size]
j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```
assume an array of $N$ integers: find all positions where $value > x$

```java
res = new array[data.size]

j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if only 1% qualifies?
assume an array of $N$ integers: find all positions where $value > x$.

```java
res = new array[data.size]

j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if only 1% qualifies?
assume an array of $N$ integers:
find all positions where $value > x$

```java
res = new array[data.size]
j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if only 1% qualifies?

memory

- data
- res
assume an array of $N$ integers: find all positions where $value > x$

```java
res = new array[data.size]
j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if only 1% qualifies?

memory
assume an array of $N$ integers: find all positions where $value > x$

```java
res = new array[data.size]

j = 0
for (i=0; i<data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if only 1% qualifies?

memory

data
assume an array of $N$ integers: find all positions where $\text{value} > x$

```java
res = new array[data.size]
j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if only 1% qualifies?
assume an array of $N$ integers: find all positions where $value > x$

```java
res = new array[data.size]
j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if only 1% qualifies?
assume an array of $N$ integers: find all positions where $value > x$

```java
res = new array[data.size]
j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if only 1% qualifies?
assume an array of $N$ integers: find all positions where $value > x$

```java
res = new array[data.size]
j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if only 1% qualifies?
assume an array of $N$ integers: find all positions where $value > x$

```java
res = new array[data.size]

j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if only 1% qualifies?
assume an array of $N$ integers: find all positions where $\text{value} > x$

```
res = new array[data.size]
j = 0
for (i=0; i<data.size; i++)
    if data[i] > x
        res[j++] = i
```
assume an array of $N$ integers: find all positions where $value > x$

```java
res = new array[data.size]

j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if only 1% qualifies?

but how can we know?
assume an array of $N$ integers: find all positions where $\text{value} > x$

$$\text{res}=\text{new array}[\text{data.size}]$$

$$j=0$$

for ($i=0; i<\text{data.size}; i++$)
  if $\text{data}[i] > x$
    $\text{res}[j++] = i$

what if 90% qualifies? result size = qualifying values $\times x$ bytes
assume an array of \( N \) integers: find all positions where \( \text{value} > x \)

\[
\text{res} = \text{new array}[\text{data.size}]
\]

\[
j = 0
\]

\[
\text{for (i=0; i<\text{data.size}; i++)}
\]

\[
\text{if data}[i] > x
\]

\[
\text{res}[j++] = i
\]

what if 90% qualifies?

result size = qualifying values * x bytes

bit vector for \( \text{res} \)?

\[
\begin{align*}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0
\end{align*}
\]

vs

\[
\begin{align*}
1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1
\end{align*}
\]
assume an array of $N$ integers: find all positions where $value > x$

```java
res = new array[data.size]

j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if 90% qualifies?

result size = qualifying values * x bytes

bit vector for res?

if statements = bad, bad, bad
assume an array of $N$ integers:
find all positions where $\text{value} > x$

and we haven’t even started discussing
about how to find the qualifying values…

```java
res = new array[data.size]
j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

what if 90% qualifies?

result size = qualifying values * $x$ bytes

bit vector for res?

```
1
0
0
0
1
0
1
1
1
0
1
0
1
```

if statements = bad, bad, bad
assume an array of $N$ integers: find all positions where $\text{value} > x$

```java
res = new array[data.size]
j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

thread1, core1
thread2, core2
thread3, core3
...

logically partition
assume an array of $N$ integers: find all positions where $value > x$

```java
res = new array[data.size]

j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

logically partition

thread1, core1
thread2, core2
thread3, core3
...

NUMA architectures?
SIMD functionality?
& what about result writing?

not as simple as spinning off N threads...
assume an array of $N$ integers: find all positions where $\text{value} > x$

\[
res = \text{new array}[\text{data.size}]
\]

\[
j = 0
\]

\[
\text{for } (i=0; i<\text{data.size}; i++)
\]

\[
\hspace{1cm} \text{if data}[i] > x
\]

\[
\hspace{1cm} \text{res}[j++] = i
\]

$N \gg 1$ queries in parallel

$q1,q2,q3$
assume an array of \( N \) integers: find all positions where \( \text{value} > x \)

```java
res = new array[data.size]

j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

\( N >> 1 \) queries in parallel

q1, q2, q3
assume an array of $N$ integers: find all positions where $value>x$

res=new array[data.size]

j=0

for (i=0; i<data.size; i++)
    if data[i]>x
        res[j++] = i

$N>>1$ queries in parallel
assume an array of $N$ integers: find all positions where $value > x$

```java
res = new array[data.size]

j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i
```

$N \gg 1$ queries in parallel
assume an array of $N$ integers: find all positions where $value > x$

res = new array[data.size]

j = 0
for (i = 0; i < data.size; i++)
    if data[i] > x
        res[j++] = i

$N \gg 1$ queries in parallel
res=new array[10]  

j=0  
for (i=0; i<10; i++)  
  if data[i]>x  
    res[j++]=i  

vs  

res=new array[10]  
j=0  
if data[0]>x res[j++]=i  
if data[1]>x res[j++]=i  
if data[2]>x res[j++]=i  
if data[3]>x res[j++]=i  
if data[4]>x res[j++]=i  
if data[5]>x res[j++]=i  
if data[6]>x res[j++]=i  
if data[7]>x res[j++]=i  
if data[8]>x res[j++]=i  
if data[9]>x res[j++]=i
assume an array of \( N \) integers: find all positions where \( value > x \)

option 1: **scan** all data

option 2: use a **tree** (do not consider tree generation costs)

which one is best
cost: data touched & computation

![Graph showing cost vs time with speed, cpu, and mem as parameters.](image-url)
design

- **logical** design
- **physical** design
- **system** design
how can I prepare?

1) start browsing some basic texts

**Get familiar with the very basics of traditional database architectures:**

**Get familiar with very basics of modern database architectures:**

**Get familiar with the very basics of modern large scale systems:**

2) play with basic data structures
implementation in C (linked list/hash table/tree)
**Action steps:**

1) Read the syllabus & website carefully,
2) Register to Piazza,
3) Do P0 if you have not taken CS165 and check self-test,
4) Register for paper presentation (week 2),
5) Start submitting your paper reviews (week 3)

---

**web site:** http://daslab.seas.harvard.edu/classes/cs265/
**piazza:** piazza.com/harvard/spring2018/cs265/home
**office hours:** Stratos: Wed/Thur/Fri, 3-4pm, MD139
**TF office hours:** check website
**textbook:** nope
research papers will be available from the Harvard network
data systems 101
BIG DATA SYSTEMS
prof. Stratos Idreos

next week

modern main-memory
optimized data systems
&
semester projects