DBMS Metrology : Measuring Query Time

Sabah Currim, Richard T. Snodgrass, Young-Kyoon Suh,
Rui Zhang, Matthew Wong Johnson, Cheng Yi

Problem

❑ Varying measured time of ten executions of the below query on PostgreSQL

```
SELECT t0.id3, t1.id2
FROM ft_HT3 t3, ft_HT1 t1, ft_HT3 t2, ft_HT1 t0
WHERE t3.id3=t1.id2 AND t1.id2=t2.id1
      AND t2.id1=t0.id4
```

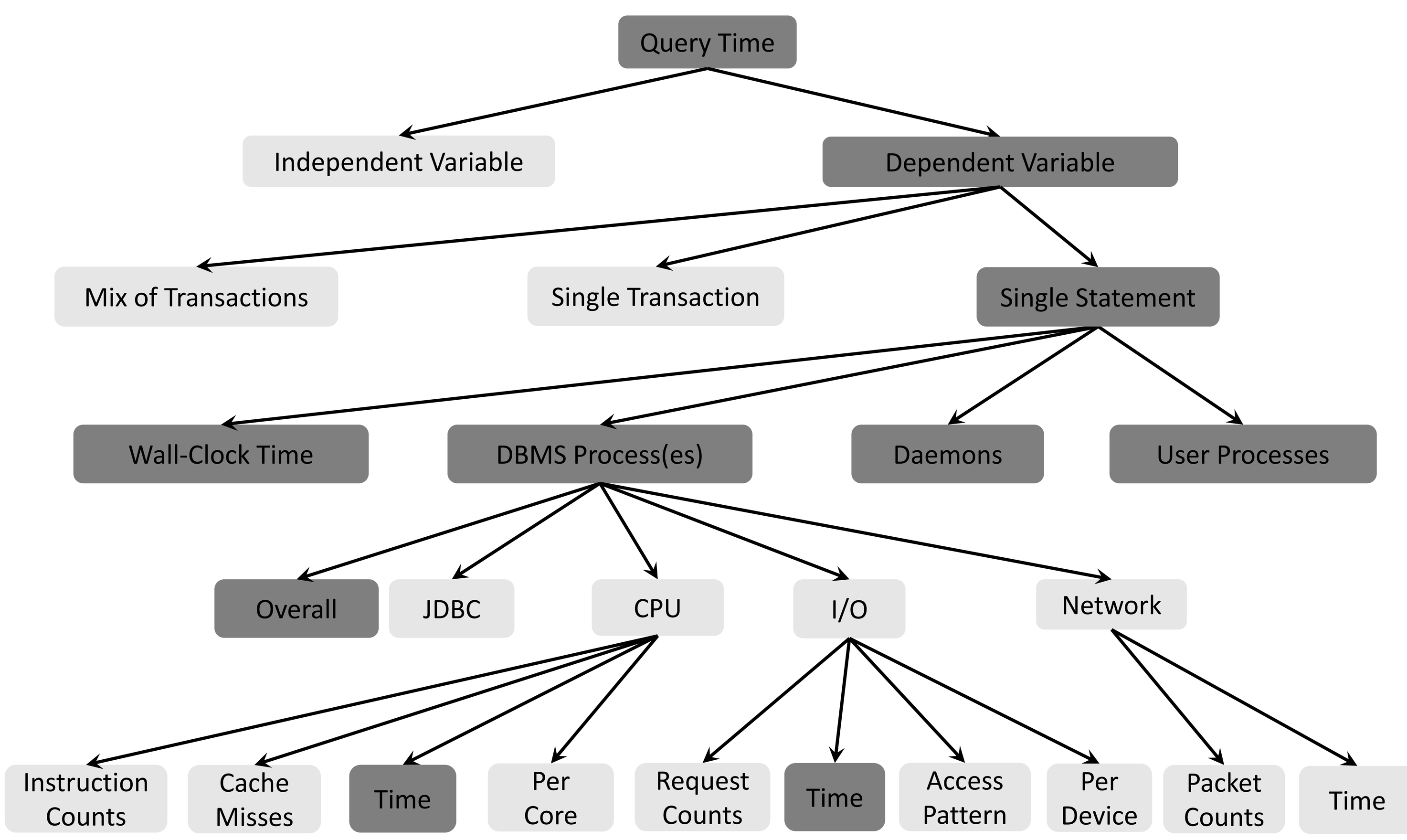
(`ft_HT1` : variable table contains 177,700 tuples, each with four integers

`ft_HT3` : fixed table with 2M rows, each with four integers)

	1	2	3	4	5	6	7	8	9	10	Avg	Std Dev
$Time_{meas}$ (msec)	6530	6571	8764	7961	8427	7829	8246	8506	8239	6991	7806	818

❑ How to measure in a more accurate and precise manner query execution time?

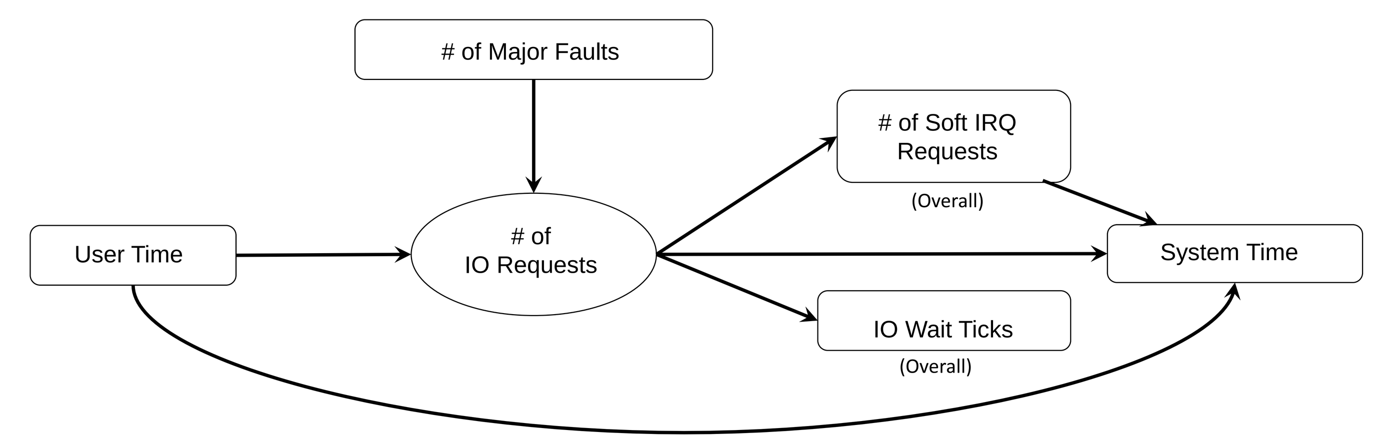
Taxonomy



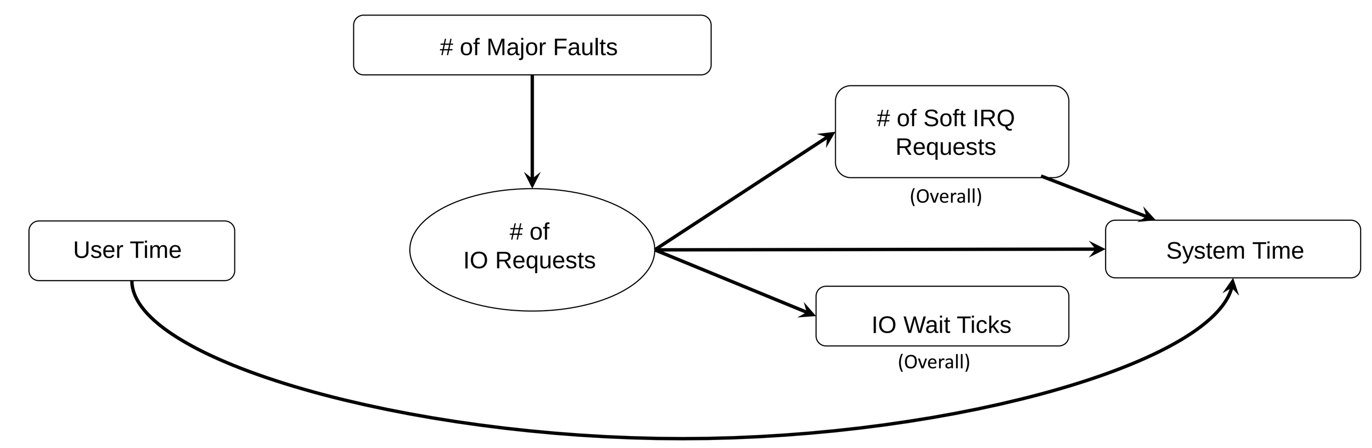
Measures

- ❑ Wall-Clock Query Time Measurement
  - ❑ Measurement Steps
    - (a) Stopped as many OS daemons as possible
    - (b) Eliminated network delays
    - (c) Eliminated user interactions
    - (d) Ensured repeatability of environment
    - (e) Ensured repeatability of I/O
    - (Steps (a)-(c) improve accuracy while (d) - (e) address precision.)
- ❑ Per-Process Measures
  - ❑ minflt : # of minor page faults
  - ❑ majflt : # of major page faults
  - ❑ utime : # of ticks in which a process was running in user mode
  - ❑ stime : # of ticks in which a request from the process was being handled by OS
  - ❑ guest time : # of ticks in spent running a virtual CPU for a guest OS
- ❑ Overall Measures
  - ❑ utime : # of ticks in which a user process was executing
  - ❑ user mode with low priority
  - ❑ stime : # of ticks in which the OS was servicing a system call or interrupt
  - ❑ idle time : # of ticks when the processor has nothing to do
  - ❑ IOwait time : # of ticks in which the system had no processes to run due to I/O waiting
  - ❑ irq : (interrupt requests) handled by the system
  - ❑ softirq : # of soft interrupt requests
  - ❑ steal time : # of ticks spent in other operating systems when running in a virtualized environment
  - ❑ processes : # of forks

Structural Causal Model



< Model for the DBMS Query Process >



< Model for Non-Query (Utility and Daemon) Process >

Timing Protocol

Step 1) Perform sanity checks

✓Overall

- ❑ # of Missing Queries : queries not executed
- ❑ # of Process Info Failures : query executions that do not have process information
- ❑ # of Unique Plan Violations : queries having different plans at the same cardinality

✓ Query Executions (QEs)

- ❑ % of DBMS Time Violations : pct. of QEs with (the total DBMS time) < (the total daemon process time)
- ❑ % of Zero Query Time Violations: pct. of QEs % zero query time
- ❑ % of No Query Process Violation(s) : pct. of QEs having no query process
- ❑ % of Other Query Process Violation(s) : pct. of QEs having other query processes
- ❑ % of Other Utility Process Violation(s) : pct. of QEs having other utility processes

✓Query-at-Cardinalities (Q@Cs)

- ❑ Excessive Variation in Query Time : pct. of Q@Cs with (std. of query time) > 20% × (avg. query time)
- ❑ Strict Monotonicity Violations : pct. of a pair of Q@Cs with (the query time at a lower cardinality) > (the query time at a higher cardinality)
- ❑ Relaxed Monotonicity Violations : pct. of a pair of Q@Cs with (the query time at a lower cardinality - std. of the query time) > (the query time at a higher cardinality + std. of the query time)

Step 2) Drop QEs

(i) Failing to pass sanity checks

(ii) Having a *stopped* or *phantom* process

(iii) Having two times IOwait ticks bigger than median IOWait ticks > 0

(iv) Having non-zero IOwait ticks from DB2 and non-zero IOwait ticks > 1% * (query user time)

Step 3) Drop Q@Cs

(i) Having no identified query process

(ii) Having average measured time less than 20ms

(iii) Having less than valid six Qes

Step 4) Calculate Query Times

	1	2	3	4	5	6	7	8	9	10	Avg	Std Dev
$Time_{meas}$ (msec)	6530	6571	8764	7961	8427	7829	8246	8506	8239	6991	7806	818
SU_{query} (ticks)	455	454	458	457	—	453	456	460	460	—	457	2.6
$MajFlt_{Qdbms}$ (msec)	0	0	0	0	—	0	0	0	0	—	0	0
$IOwait_{meas}$ (ticks)	150	149	349	269	—	264	301	275	258	—	252	69.5
$IOwait_{calc}$ (ticks)	74.9	75.9	76.1	74.1	—	74.3	74.6	73.3	75.1	—	74.8	0.9
$Time_{calc}$ (msec)	5298.5	5298.9	5341.5	5310.7	—	5293.3	5305.9	5333.0	5311.1	—	5311.6	17.1

<Final (median) $Time_{calc}$: 5308.30 (msec)>

Step 5) Perform post-sanity checks

- ❑ Excessive Variation in Query Time
- ❑ Strict Monotonicity Violations
- ❑ Relaxed Monotonicity Violations

Predicted Correlations

	a	b	c	d	e	f	g	h	i	j	k
I	query UTime/ query STime high	query I/O req/ query STime high	query UTime/ query I/O req high	query MajFlts/ query I/O req low	query MajFlts/ query STime low	utility MajFlts/ utility I/O req low	utility MajFlts/ utility STime low	utility I/O req/ utility STime high	daemon I/O req/ daemon I/O req low	daemon MajFlts/ daemon STime low	daemon I/O req/ daemon STime high
II	query I/O req/ overall IO wait med	query I/O req/ overall Soft IRQ med	query Soft IRQ/ query STime med	utility I/O req/ overall I/O wait low	utility STime/ overall Soft IRQ low	daemon I/O req/ overall I/O wait low	daemon STime/ overall Soft IRQ low	utility I/O req/ overall Soft IRQ med	daemon I/O req/ overall Soft IRQ med		
III	overall MajFlts/ overall I/O req low	overall I/O req/ overall STime high	overal IO req/ overall I/O wait low	overall I/O req/ overall Soft IRQ med	overall Soft IRQ/ overall STime med						
IV	query UTime/ overall Soft IRQ med	query UTime/ overall IO wait med	query MajFlts/ overall Soft IRQ med	query MajFlts/ overall IO wait med	utility MajFlts/ overall Soft IRQ low	utility MajFlts/ overall IO wait low	daemon MajFlts/ overall Soft IRQ low	daemon MajFlts/ overall IO wait low			
V	overall UTime/ overall Soft IRQ med	overall UTime/ overall IO wait med	overall MajFlts/ overall Soft IRQ med	overall MajFlts/ overall IO wait low	overall MajFlts/ overall STime low						
VI	query UTime/ overall UTime high	query STime/ overall STime high	query IO req/ overall I/O req high	query MajFlts/ overall MajFlts low	utility MajFlts/ overall MajFlts med	daemon MajFlts/ overall MajFlts med					
VII	overall IO wait/ query STime med										

Testing the Causal Model

❑ Exploratory Analysis

❑ Did correlational analysis on a small portion of the query runs

❑ Examined our assumptions against the analysis results and resulted in the two-part model through refinement

❑ Resulted in a set of 27 correlations, each with an expected level : *low* (< 0.3), *medium*, or *high* (≥ 0.7)

❑ Not involving the latent variable (# of IO requests) out of 45 expected correlations

❑ Confirmatory Analysis

❑ Found that the level predicted by our model either exactly matched that of the actual level

❑ Produced only ***eleven*** that were of concern, none of which presents a serious challenge to the model for the *108* testable (*27* interactions for each of four DBMSes)

❑ Reduced these interactions of concern (to three) dramatically on the refined data through the timing protocol

Future Work

- ❑ Timing protocol refinement by
  - ❑ Incorporating network delays for a remote disk
  - ❑ Utilizing block read and write statistics available from the DBMSes and bytes read and written from the O/S
  - ❑ Accommodating multiple disks, connected by a single or distinct channels,
  - ❑ Accommodating multiple processor cores
  - ❑ Accommodating phantom processes while eliminating their impact on the computed time
  - ❑ Extending PostgreSQL to clear its cache
  - ❑ Ensuring repeatability of file fragmentation
  - ❑ Supporting the Windows operating system, which has different per-process metrics, and thus might require an altered causal model and a different regression model and calculation of query time
  - ❑ Accommodating multiple disks
  - ❑ Measuring single transactions that incorporate multiple statements
  - ❑ Measuring a mix of transactions

THE UNIVERSITY OF ARIZONA.

ACM SIGMOD NEW YORK, USA, 2013

NSF