Coding for Network Performance

Tom Bascom, White Star Software

Thursday 11:45-12:45

Abstract: Are you wondering why your client/server code is so slow? And what you can do about it?

This session will discuss the OpenEdge client/server messaging protocol and its impact on the performance of database queries. We will cover coding best practices, tuning opportunities, testing methodologies and present benchmark results!

```
FOR EACH customer FIELDS ( name balance ) NO-
LOCK:
  /* ... */
END.
```

Coding for Network Performance

Tom Bascom, White Star Software
tom@wss.com

White Star Software

Agenda

- What Are Network Messages?
- Improving Existing “Queries”
- FIND FIRST Does Not Help
- How to Tune Network Messages
- Summary

What Are Network Messages?

Progress Network Messages

- When Progress runs over a network data is marshalled into “messages”:
  - The client requests data (makes a query with FIND or FOR EACH…)
  - The server resolves the query and sends back the results
  - The client might upgrade locks
  - The client eventually releases locks and ”cursors”
- “Messages” are potentially broken up by TCP/IP into multiple “packets”.
- A single round trip may only take a few milliseconds – but they add up very quickly.
- During peak periods some applications generate 200,000 or more messages per second!

FIND NO-LOCK

```
FIND customer WHERE cust-num = 10 NO-LOCK.
```

One round trip (two messages) to retrieve the record.

Client	---> Server	Requests record
Server	---> Client	Sends record back

One more one way message to release the cursor.

Client	---> Server	Releases the cursor

FOR EACH NO-LOCK

```
FOR EACH customer NO-LOCK:
```

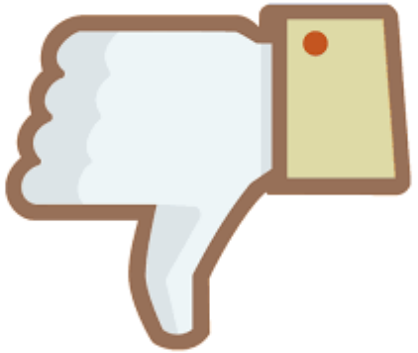
Fetch the first record with one round-trip then as many records as can fit in a message per round-trip. Subject to the following limits:

`-Mm`	message buffer size (default 1024)
`-prefetchDelay`	skips the single record initial message (default disabled)
`-prefetchFactor`	% full to fill the message (default 0)
`-prefetchNumRecs`	number of records per message (default 16)

A FOR EACH is potentially many fewer messages than the equivalent set of FIND statements. Could return dozens or even hundreds of records in a single round trip!

FOR EACH, With JOIN & SORT

```
FOR EACH customer NO-LOCK,
    EACH order NO-LOCK OF customer
      BY ship-date:
```



Performance depends on if the sort is only on the main table or not.

This process can use a lot of round trips!

Not recommended, try to use nested FOR EACH statements instead:

```
FOR EACH customer NO-LOCK:
  FOR EACH order NO-LOCK OF customer:
    /* would need to resolve ordering issue… */
```

Or, better-yet, build temp-tables (more on this later)

Improving Existing Queries

Pick Your Battles

The performance enhancement possible with a given improvement is limited by the fraction of the execution time that the improved feature is used.

-- Amdahl's Law

$$S = \frac{T_s}{T} = \frac{1}{f_p / n + f_s}$$

In other words:

- Trying to improve small things that nobody notices probably isn't the road to fame and fortune.

- Big queries that return lots of data and which are frequently used by lots of users will be much more noticeable.

Simple Things That Help Existing Queries

- FIELDS – can reduce the size of records and thus allow more records to be packed into a message, i.e.

```
FOR EACH customer FIELDS ( name balance ) NO-LOCK:
```

or:

```
FOR EACH customer EXCEPT ( photo ) NO-LOCK:
```

- CACHE – client side buffer for records returned (default 50 - supposedly)

```
DEFINE QUERY q FOR customer FIELDS ( name ) CACHE 1000
```

Things That Hurt (A Lot)

- SHARE-LOCK (frequently accidental)
  - Breaks the bundling of records
  - For a large result set SHARE-LOCK might be literally 100x worse!
  - Easy to accidentally get wrong
- Bad WHERE Clauses
- Nesting and JOINs
- CAN-DO()
  - Is a SECURITY function – not a string function
  - Always evaluated on the client
  - Has many unexpected behaviors

Pointless Waste of Programmer Time

- FIND FIRST

FIND FIRST Does Not Help

FIND FIRST (and LAST)

- Reflexive and automatic use on each and every FIND does NOT improve your code.

- It is NOT a “standard”.
- Nor is it a “best practice”.
- Nor does it “always work”.

- Yes, I know it is all over the place in certain code.

Unique FINDs

- FIND is designed to return exactly one or zero records.
- 99.44% of FIND statements should be for UNIQUE records.
- This is one of Progress' big advantages over SQL.
- If the WHERE clause specifies a **unique** record then FIRST adds no value.
- Worse – it confuses the maintenance programmer by implying that there /should/ be an ordered result-set.

Unique FIND FIRST performance

- It is NOT faster.
- It does NOT “eliminate a check for ambiguous records”.

```
FIND FIRST customer NO-LOCK.
FIND customer NO-LOCK WHERE custNum = 1.
FIND FIRST customer NO-LOCK WHERE custNum = 1.
```

- All of the statements above take the same time to run and have the same “logical” impact on the db engine.
- All statements execute the same number of “logical IO ops” (ProTop, PROMON or VST “block access”).
- Feel free to test it yourself!

Faster FIND with FIRST?

- But what if FIRST does actually make a query faster?
  - You have not specified UNIQUE criteria!
  - You are missing an appropriate index to match your WHERE clause.
  - Maybe your WHERE clause isn't doing what you think it should be doing?

FIND FIRST Slogan

Returning the Wrong Record Faster!

A program that produces incorrect results twice as fast is infinitely slower.

— John Osterhout

FIND SECOND?

- You used FIND FIRST anyway… what are you doing about the second record?

- If there actually is a second record and you are actually doing something with it:
  - How did you specify the ordering?
  - If you don't care about order – what does FIRST mean?
  - Are you treating it exactly the same as the FIRST record from a 3NF perspective?
  - Are you processing the entire result set? **Why didn't you use FOR EACH?**

“It Always Works”

- This usually means that the programmer does not want to deal with:

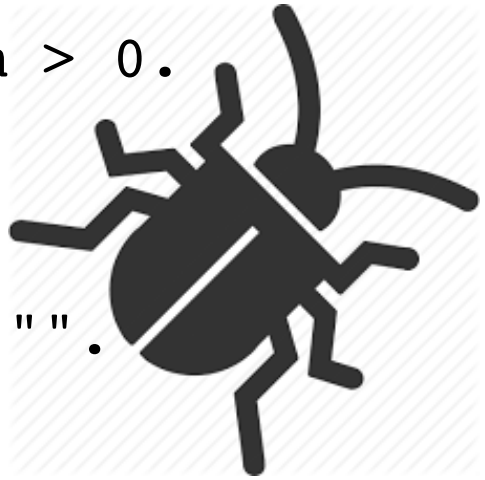
```
More than one Customer records found by a unique FIND. (3166)
```

- Adding FIRST will “make it go away”.
- It also means that your result is potentially wrong:
  - What if you forgot a component of the index?
  - Or didn’t know that a previously unused feature has been enabled by the users?
  - Or the users suddenly create a second magical record?

Magic FIRST Records

- Records that are special by convention.
- No specific attribute identifies the usage.
- A clear violation of Third Normal Form.

```
find first customer no-lock where custNum > 0.
display custNum name discount.
defaultDiscount = discount.

find first customer no-lock where name > "".
display custNum name discount.
defaultDiscount = discount.
```



FIND FIRST Summary

- FIND FIRST is almost always a sign of lazy programming
- It does not improve performance
- It can create bugs and mask existing bugs
- Some code is infested with it – but there is no reason to make the problem worse by continuing the habit

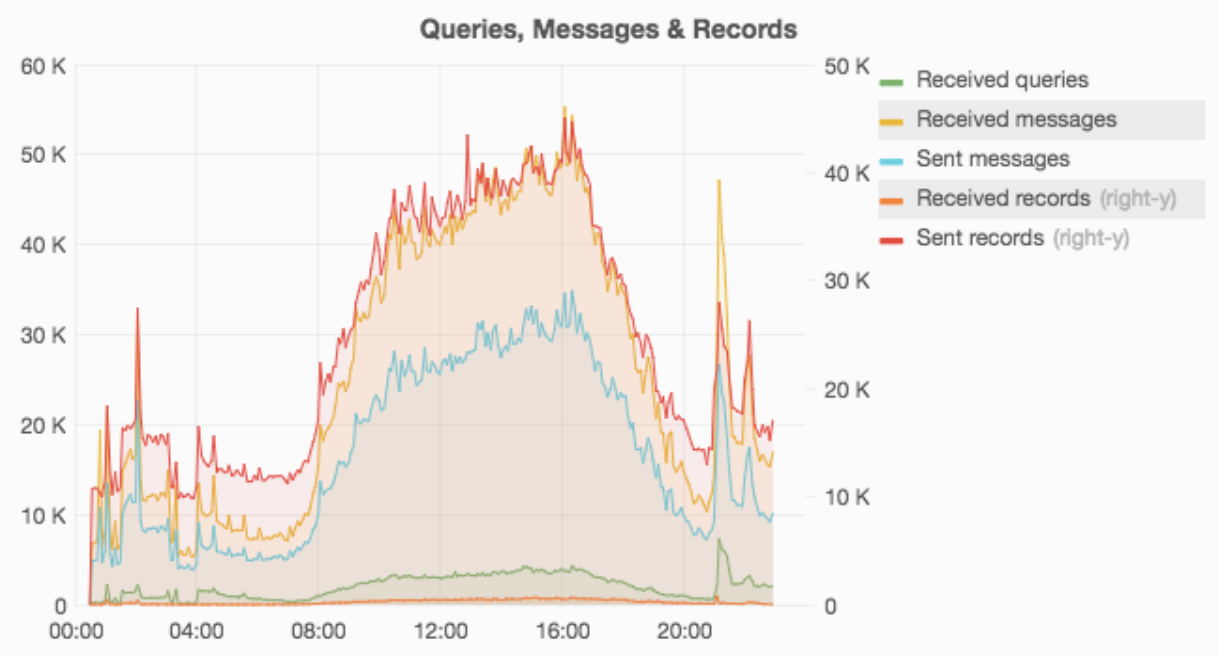
How to Tune Network Messages

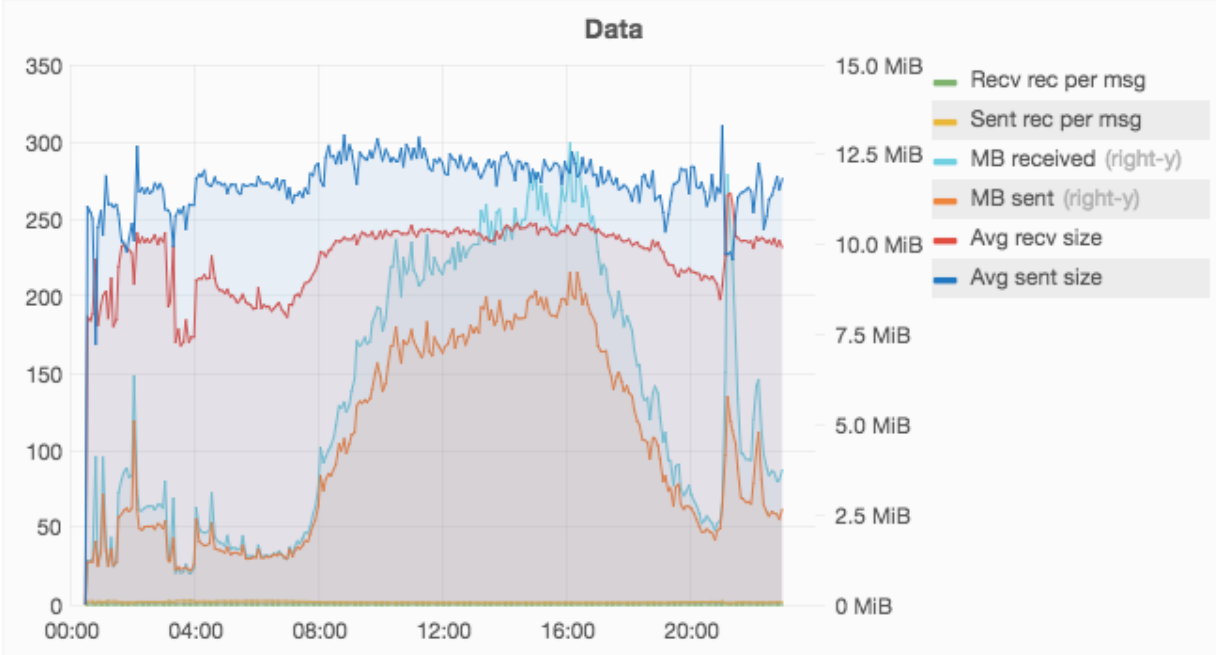
What to Look For in VSTs/ProTop/PROMON

- “Records per Query”

```
for each _field fields( _field-name ) no-lock:
end.

find _actServer no-lock where _Server-id = 2.        /* if we are the sole user… */

display
  _Server-QryRec
  _Server-recSent
  ( _Server-recSent / _Server-QryRec )
.

    Queries received          Records sent
_____________________ _____________________

                   22                  3832     174.18
```

ProTop Portal





(A new “records per query” is coming in the next release, for now you can manually calculate it from “received queries” and “sent records”.)

White Star Software

ProTop ChUI

```
xyzzy   Auto Interval Rate JSON 226461 0 0.518                    ProTop Version 3.3sx                                                    2017/11/09
13:50:13
xyzzy   0 0                                                                    /db/xyzzy
         Hit%       99.99         Commits:     341       Examined:   7532      APW Writes:     888              DB UpTime   49d 20:29    Connections:
1803
   Log Reads:     2464995           Undos:   13948         New RM:   7051       APW Write%     100              Backup Age      12:35           -n %
60%
    OS Reads:         347    Lock Tbl HWM:  278422        From RM:   7051    Bufs Scanned:    3338                                           Brokers:
10
   Rec Reads:     1037415    Curr # Locks:    1240      RM Locked:   6967   APW Scan Wrts:       2              Oldest TRX:   01:28:47    4gl Servers:
  100
 LogRd/RecRd:        2.38        Lock Tbl%   0.12%      From Free:      0      APW Q Wrts:       0            Curr BIClstr:      92414    SQL Servers:
22
  Log Writes:       30139   Modified Bufs:    5858       Front2Bk:    481    Chkpt Q Wrts:     887             Old BIClstr:      92374    4gl Clients:
1620
   OS Writes:         888    Evicted Bufs:       0                             Flushed Bufs:       0            Num BIClstrs:         40    SQL Clients:
10
 Rec Creates:        7050                                                        Chkpt Len:                     BI MB Used:       2560     App Server:
43
 Rec Updates:         152                                                                                                                  Web Speed:
    0
 Rec Deletes:           7                                                                          BI       AI     Curr AI Ext:    1 of 12   BIW/AIW/WDOG:
1 1 1
   Rec Locks:      247647                                                             Notes:    72627    72627      Curr Seq#:       6217        AI Mgmt:
    1
 Idx Blk Spl:           0                                                       BI/AI Write%       65      100       Empty AI:         11           APWs:
    4
                                                                              Writes to Log:      371      354        Full AI:          0          Local:
1597
   Rec Waits:           0     IO Response:    0.14                            BI/AI Writes:      241      353                                   Remote:
10
 Resrc Waits:          26       BogoMIPS:    5.35                              Partial Wr:       17        0      Locked AI:          0          Batch:
23
 Latch Waits:         101       ZippySHM:    4.00                              Busy Waits:       23        0      pica Used:          0            TRX:
```

Test Harness

```
/* foreach.p */

{actsrv_hdr.i}
{actsrv_init.i}

for each _field no-lock:                                    /* test 1 */
end.

{actsrv_end.i "for each"}
{actsrv_init.i}

for each _field fields( _field-name ) no-lock:              /* test 2 */
end.

{actsrv_end.i "for each fields()"}
```

Test Harness

```
/* actsrv_hdr.i */

define variable msgRecv   as integer no-undo.
define variable msgSent   as integer no-undo.
define variable recSent   as integer no-undo.
define variable qryRecv   as integer no-undo.

for each _field no-lock:                    /* ensure that _field is in -B */
end.
```

Test Harness

```
/* actsrv_init.i */

find _actServer no-lock where _Server-id = 2.
assign
  msgRecv   = _Server-msgRec
  msgSent   = _Server-msgSent
  recSent   = _Server-recSent
  qryRecv   = _Server-QryRec
.

etime( yes ).
```

Test Harness

```
/* actsrv_end.i */

find _actServer no-lock where _Server-id = 2.
assign
  msgRecv   = _Server-msgRec    - msgRecv
  msgSent   = _Server-msgSent   - msgSent
  recSent   = _Server-recSent   - recSent
  qryRecv   = _Server-QryRec    - qryRecv
.

output to value( "nettraffic.txt" ) append.
file-info:file-name = "nettraffic.txt".
if file-info:file-size < 10 then
  do:
    put unformatted "    totMsgs    msgRecv… " skip.
    put unformatted "----------- ----------… " skip.
  end.
```

Test Harness

```
put
  ( msgRecv + msgSent )   format ">>>,>>>,>>9"
  qryRecv                 format ">>>,>>>,>>9"
  recSent                 format ">>>,>>>,>>9"
  ( recSent   / qryRecv ) format ">>>,>>>,>>9"
  etime                   format ">>,>>9"
.

put unformatted " " trim( session:parameter + " {1}" ) skip.

output close.
```

Demo!

Impact of Message Size & Prefetch Options

```
for each _index fields(_field-name) no-lock:
end.
```

totMsgs	qryRecv	recSent	recs/qry	etime	net time	Description...
198	97	1758	18	10	208	-Mm 1024
208	102	1758	17	14	222	-Mm 4096
192	94	1758	19	9	201	-Mm 8192
180	88	1758	20	11	191	-Mm 16384
162	79	1758	22	14	176	-Mm 32600
152	74	1758	24	8	160	-prefetchDelay
154	75	1758	23	12	166	-prefetchDelay -prefetchFactor 100
8	2	1758	879	22	30	-prefetchDelay -prefetchFactor 100 -prefetchNumRecs 10000

Impact of Message Size & PrefetchNumRecs

`-prefetchNumRecs` ***dominates!***

totMsgs	qryRecv	recSent	recs/qry	etime	net time	Description...
102	49	1758	36	8	110	-Mm 1024 -prefetchNumRecs 10000
28	12	1758	147	9	37	-Mm 4096 -prefetchNumRecs 10000
16	6	1758	293	13	29	-Mm 8192 -prefetchNumRecs 10000
10	3	1758	586	9	19	-Mm 16384 -prefetchNumRecs 10000
8	2	1758	879	23	31	-Mm 32600 -prefetchNumRecs 10000
8	2	1758	879	22	30	-prefetchDelay -prefetchNumRecs 10000
146	71	1758	25	13	159	-prefetchDelay -prefetchFactor 100
8	2	1758	879	22	30	-prefetchDelay -prefetchFactor 100 -prefetchNumRecs 10000

Impact of Basic Coding Approaches

totMsgs	qryRecv	recSent	recs/qry	etime	net time	Description...
3519	0	1758	?	139	3658	do while … find no-lock
5276	0	1758	?	174	5450	do while … find share-lock
28	12	1758	147	7	35	for each no-lock
8	2	1758	879	22	30	FENL fields()
5277	1758	1758	1	155	5432	FE share-lock
3520	1758	1758	1	134	3654	FE exclusive-lock
3519	1758	1758	1	87	3606	open query
3519	1758	1758	1	84	3603	open query fields()
7	2	1758	879	29	36	open query fields() cache 50
7	2	1758	879	30	37	open query fields() cache 5000
20	8	1758	220	113	133	sql89 select *
8	2	1758	879	37	45	sql89 select _field-name

No, I am not endorsing SQL89

Nesting and Joins

```
/* simple nesting */

for each _file no-lock:
  for each _field no-lock of _file:
  end.
end.
```

```
/* ugly sort criteria */

for each _file  no-lock,
    each _field no-lock of _file
    by _field-name:
end.
```

```
/* use a join instead of nesting */

for each _file  no-lock,
    each _field no-lock of _file:
end.
```

totMsgs	qryRecv	recSent	recs/qry	etime	net time	Description...
587	195	1950	10	33	620	nested FE
587	195	1950	10	33	620	joined FE
6183	195	4748	24	207	6390	joined FE w/ sort on inner field
33	14	1951	139	150	183	TT option

Nesting and Joins – TT Option

```
define temp-table tt_file no-undo like _file
  field xdbRecid as recid
  index xdbRecid-idx is unique xdbRecid.
define temp-table tt_field no-undo like _field.

for each _file no-lock:
  create tt_file.
  buffer-copy _file to tt_file.
  xdbRecid = recid( _file ).
end.
for each _field no-lock:
  create tt_field.
  buffer-copy _field to tt_field.
end.

for each tt_file no-lock,
    each tt_field no-lock where tt_file.xdbRecid = tt_field._file-
recid
  by tt_field._field-name:
end.
```

A major reduction in network traffic and a big improvement in “across the network time”!

WARNING - don’t just do this automatically. If your code is designed to be run with shared memory this approach may make it slower.

Use App Servers

- Position app servers “close” to the db server to minimize traffic “over the wire”
- In a virtualized environment VMs on the same physical server may implement TCP/IP in memory – avoiding NICs completely
- App server results are “streamed” to the client – they are not grouped as records

Summary

Summary

- Large –prefetchNumRecs is critical
- Large –Mm message sizes are helpful
- Where feasible the FIELDS phrase is very helpful
- Use temp tables to cache data locally!
- Small differences in coding approaches can make a big difference
- Simple mistakes can be catastrophic!

Questions?

Thank You!

Kbase 18342

Every FIND, FOR EACH, OPEN QUERY, GET, PRESELECT, ASSIGN, CREATE, DELETE (all data manipulation statements) generates client/server traffic when the database is connected with the -H -S client connection parameters. Being aware of the above and knowing how much traffic each 4GL/ABL statement generates can make the difference in a fast versus a very slow networked application.

SIMPLE FIND
FIND customer WHERE cust-num = 10 NO-LOCK.

No locking is involved.
One round trip to retrieve the record.
One more one way message to release the cursor.

Client ---> Server Requests record
Server ---> Client Sends record back
Client ---> Server Releases the cursor

GENERIC FIND
One round-trip to request record/get the record
No extra message to request a lock (RECID finds don't allocate a cursor).
If the cursor will not be used again it is released immediately, if it might be used later it will only be released when the record goes out of scope which is done with a one way message.
SHARE-LOCKs are released with a one way message.
In a transaction no locks are released until the end of the transaction.