Technical Note 04-33
Execute on server

By Bertrand Soubeyrand, 4D Developer.
Technical Note 04-33

（原題: Taking Advantage of Execute on Server）

概要

クライアント/サーバで 4D を運用する場合、コードが最適化されていないとネットワークに多大な負担をかかってしまうかもしれません。代表的なのが、クエリやフォーミュラによる更新を巨大なセレクションに対して実行する場合です。この Tech Note では、ストアドプロシージャの適切な使用法について論じています。

一部のコマンドはクライアント/サーバ用に最適化されており、スタンドアロン時とは異なる動作をしますが、すべてのコマンドでそうであるわけではありません。最適化されていないコマンドは、クライアント/サーバに移行した途端、著しくパフォーマンスが低下します。

Execute on server と New process は動作が似ていますが、プロセスを立ち上げる場所が異なっています。Execute on server は新規プロセスをサーバ側で立ち上げますが、New process はクライアントマシンでプロセスを立ち上げます。スタンドアロンでは、どちらも同じ動作をします。

Execute on server コマンドは、新規プロセスを起動させて別メソッドをコールするような使用法が考えられます。

```
  `method  LaunchProcess
$I_Process:=Execute on server("TaskInProcess";1024*64;"ProcessName")

`method  TaskInProcess
ALL RECORDS([Test])
QUERY  SELECTION([Test])
```

もっと精錬されているのは、再帰メソッド、つまり自らをコールするようなメソッドです。

再帰メソッド

再帰メソッドでは、パラメータ数による条件分岐で外部からコールされたのか、自らをコールしたのかを見分けます。外部からコールされた場合は Execute on server で自らをコールします。自らコールされた場合のルーチン(BRANCH B)はサーバで処理されます。

クライアント側のルーチン(BRANCH A)は、このままでは終了してしまうので、GET PROCESS VARIABLE でサーバからの返答を待ちます。

```
` Variables to synchronize the process
C_BOOLEAN(B_End;B_Variable_Read)
`Result to display in A
C_TEXT(T_Time)
Case of
	¥ (Count parameters=0)
	` \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
	` BRANCH A
	` //////////////////////////////////////
	B_End:=False
	$I_Process:=Execute on server(Current method name;1024*32;"Test";"Launch")
	Repeat
		DELAY PROCESS(Current process;60)
		IDLE
		GET PROCESS VARIABLE($I_Process;B_End;B_End)
	Until (B_End)
	` Branch B has completed, retrieving the time elapsed
	GET PROCESS VARIABLE($I_Process;T_Time;T_Time)
	` Branch B is made aware that Branch A retrieved the time elapsed
	B_Variable_Read:=True
	SET PROCESS VARIABLE($I_Process;B_Variable_Read;B_Variable_Read)
	` displaying time elapsed and ends process
	ALERT("Time elapsed: "+T_Time)
Else
	` \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
	` BRANCH B
	` //////////////////////////////////////
	B_End:=False
	` === CODE TO BE RUN ===
	$L_Start:=Milliseconds
	Time_Stamp_Method
	T_Time:=(Time string((Milliseconds-$L_Start)/1000))
	B_End:=True
	` === end of code ===
	` waiting for BRANCH A to read T_Time
	` as long as B_Variable_read is set to False, Branch A can read T_Time
	` Branch A sets B_Variable_Read to True
	B_Variable_Read:=False
	Repeat
		DELAY PROCESS(Current process;3)
		IDLE
	Until (B_Variable_Read)
	` Exiting method, server process will die out
End case
```

サーバプロセスで変数 B_End が True にセットされると、数十分の一秒後にはクライアントプロセスに伝わります。クライアントが結果を受信したことをサーバ側で確認するためには BRANCH B が BRANCH A の反応を待ちます。このとき、サーバがクライアントの変数を読みにいくのではなく、クライアントがサーバ側のプロセス変数をセットしていることに注目することができます。サーバがクライアントに GET PROCESS VARIABLE をかけることはないはずです。

GET PROCESS VARIABLE は配列を扱うことができないので、配列は BLOB に変換して受け渡すと良いでしょう。あるいは配列をテーブルに保存するという方法もあります。専用のテーブルに結果を保存する手法には全クライアントがそれを参照できるという利点があるからです。

注記:
Execute on server を使用する場合は、サーバをロックしないために必要な手段を講じるようにして下さい。http://www.4d.com/docs/CMU/CMU40975.HTM

サンプルデータベース

このサンプルはクライント/サーバでの実行を想定しています。はじめに Demo メニューから Non-Optimized を選択します。1000 件のレコードに対して APPLY TO SELECTION が実行され、処理に要した時間が表示されます。

次に Optimized を選択します。同じ処理が Execute on server の再帰メソッドで実行され、明らかに速度が向上していることが確認できます。