Results of Performance Testing
Postgresql Version shef_decode_raw
For The
RFC Archive System[1]

April 14. 2006

1.0 Summary

Results indicate that the Postgresql version of shef_decode_raw application is much slower than the current Informix version (ob6). This is primarily due to two factors; 1) the difference in how the Informix Relational Database Management System (RDBMS) works (i.e. multi-threading capability) versus how the Postgresql RDBMS works, and 2) use of parameter settings as they were delivered with the Postgresql install package (i.e., no "tuning"). Other factors that may have impacted the performance are the level of RAID being used and the system hardware itself, the impacts of these factors are beyond the scope of this paper.

While beyond the scope of this study, tuning of the Postgresql engine appears to be the most likely way to significantly improve the speed of data posting to the archive database. The increased performance that tuning the Postgresql engine can provide has been examined for the IHFS DB in build ob6 at Northwest River Forecast Center; see the report "AWIPS Test Authorization Note – Test Results, POB6_OHD_A100768".

It is highly recommended that tuning of the Postgresql RDBMS be performed for the RFC Archive Database.

2.0 Introduction

2.1 Background

In build ob6 all AWIPS systems *except* the River Forecast Center (RFC) Archive System moved to Postgresql (from Informix) for the RDBMS. The RFC Archive System moves to Postgresql (from Informix) for the RDBMS in build ob7.2. It was agreed that information on the performance of Postgresql vs. Informix would be provided to the RFCs during the development phase so that RFCs would have an idea of the relative performance they could expect for different types of database-related tasks on the RFC Archive System. Phase 1 of the testing was completed in the summer of 2005; see the report "PostgreSQL V7.4.7 versus V8.0.3 For the RFC Archive System" dated August 8, 2005.

---

[1] Report written by A. Juliann Meyer, RAXUM Team Leader, Sr. Hydrologist – Data Systems, Missouri Basin River Forecast Center, Pleasant Hill, MO

The shef_decode_raw application is the program that processes the live data-feed of Standard Hydro-Meteorological Exchange Format (SHEF) messages and posts data to the suite of SHEF data value database tables. This is the main way observations and forecast data get into the RFC Archive Database.

2.2 Purpose

The purpose of this test is to provide the RFCs with information on the performance of the Postgresql version of the shef_decode_raw application as compared to the performance of the current Informix version.

During the review of the initial test results it was suggested that a couple of additional tests be done, these are: 1) perform a vacuum full on the database and then rerun test 3, and 2) collect information on how long a delete of a large number of rows can take.

3.0 Description of Test System

The testing was performed on two systems, ax1-nhdr for the ob6 Informix set-up and ax2-nhdr for the ob7 Postgresql set-up. Note that the ax1-nhdr is not identical in hardware to ax2-nhdr. The ax1-nhdr system was set-up in February/March 2005 as the operational support system for the RAXUM team. The ax2-nhdr system (the development system) is identical hardware to the rax systems in the field.

Note: The author is well aware that the testing conditions on ax2-nhdr are not ideal, as development work is still ongoing for the ob7.2 software delivery.

3.1 Hardware

ax2-nhdr

- Dedicated system, Rack mounted
- Intel Xeon 2.4GHz/400MHz
- 2 - 512MB PC2100 CL2.5 ECC DDR SDRAM RDIMM
- Ultra 320, ServeRAID-5i SCSI Controller (single channel)
- Six 73.4GB 10K rpm Ultra160 SCSI HS
- 10/100/1000 Port Ethernet Server Adapter
- Tape drive - 40/80GB DLTVS HH Int. SCSI Drive (Half-High) and Ultra 160 PCI Adapter (required for Tape device when using ServeRAID5i
- DVD Drive/Recorder - DVR-A04 Pioneer DVR (4.7gb)

ax1-nhdr

- Tower style case
- Dual Intel Xeon 1.5 GHz
- 2 Gig RAM
- No Raid
- 1 Seagate Cheetah 10K 36GB U320 68pin SCSI Hard Drive and 1 IBM Ultrastar 10K 36GB U160 68pin SCSI Hard Drive
- 10/100 Port Ethernet Server Adapter
- No tape drive
- No DVD recorder

3.2 Software

ax2-nhdr

RDBMS: Postgresql version 7.4.8. Uses cooked file space for the database. No logging and no replication.

OS: Red Hat Enterprise 4.0

ax1-nhdr

RDBMS: IBM Informix IDS 9.3 UC1. Uses cooked file space for the database. No logging and no replication.

OS: Red Hat 7.2

3.3 Database and Data-Feeds

Both ax1-nhdr and ax2-nhdr have RFC Archive Databases, but the information in the databases is not identical and the databases are of slightly different sizes. However the location and ingestfilter database tables are similar in that both contain definitions for stations in Missouri Basin RFC (MBRFC) and Mid-Atlantic RFC (MARFC) areas of responsibility. Messages for the non-live data-feed datasets utilize SHEF messages from the author's RFC Archive System (ax-krf). The live data-feed messages for the ax1-nhdr and ax2-nhdr system are for MARFC's area of responsibility. Because of the way that the live data-feed is generated, some of the messages may not be SHEF messages.

3.4 Database Cron Jobs

ax2-nhdr

Oper's cron was running one script related to updating table statistics, "vacuumdb -v -z". This script was run 6 times per day during most of the testing, the average running time was 30 minutes. Since mid-March the vacuumdb cron job was run twice each day, with an average running time of 70 minutes. The cron also runs a pg_dump of the database to disk nightly. This dump takes on average 2 hours and 15 minutes.

ax1-nhdr

Oper's cron was running two scripts related to updating table statistics, update-low is run at 1938Z daily and update-stats is run at 0738Z daily. Since there is no tape drive on ax1-nhdr, level-0 archive is not being done routinely. In addition, the script check_dpspaces was run at 1020z daily.

4.0 How Posting Works[2]

The focus of some of the tests performed was measuring the performance (speed) of the shef_decode_raw program when it posts (inserts and updates) to pseudo array tables versus single value per row tables. Descriptions of how the posting portion of the shef_decode_raw program works for pseudo array and single value per row tables are provided in this section.

In the descriptions that follow the three words RECORD, STRUCTURE, and ENTRY have the following meaning:

> RECORD - The information for a single data value parsed from a SHEF message.
>
> STRUCTURE - the data structure which holds all the information needed to update/insert a row in the database.
>
> ENTRY - a single row in a database table.

posting to a pseudo-array table

1) All data values in a SHEF message are parsed and stored individually in memory in their own data RECORD.

2) The posting function loops through all data RECORDS that are in memory one by one.

[2] Most of the information in this section was provided by one of the programmers at OHD/HL.

3) For the first data RECORD, and whenever the primary key changes, a select is made on the database to retrieve an ENTRY with the same primary key information as in the data RECORD. If there is a match, the retrieved database ENTRY is saved in a data STRUCTURE in memory, and the data value from the data RECORD is "stuffed" into the correct slot in the data STRUCTURE. When this STRUCTURE is written back to the database, it is done as an *update*. If there is not a match between the primary key of the data RECORD and an ENTRY in the database, then the data STRUCTURE in memory is initialized using the information from the current data RECORD. When this STRUCTURE is written back to the database, it is done as an *insert*.

4) For each data RECORD after that (2, 3, 4,...n), the primary key fields are checked against the data RECORD which was previously processed. If the primary key is the same, that RECORD value is "stuffed" into the correct slot in the data STRUCTURE already in memory. If the primary key of the current data RECORD is different, then the data STRUCTURE is written (*insert* or *update*) to the database. A select is then made on the database using the primary key information for the current data RECORD in memory, and the logic described in step 3) above is followed.

Note: The data STRUCTURE in memory is written to the database each time the primary key changes and when the last parsed data RECORD in memory is processed.

<u>posting to a single value per row table</u>

1) All data values in a SHEF message are parsed and stored individually in memory in their own data RECORD.

2) The posting function loops through all data RECORDS that are in memory one by one.

3) For each data RECORD, a select is made on the database to retrieve an ENTRY with the same primary key information as in the data RECORD. If there is a match, the database ENTRY retrieved is saved in a data STRUCTURE in memory, and the data value from the data RECORD in memory is "stuffed" into the correct slot in the data STRUCTURE. When this STRUCTURE is written back to the database, it is written as an *update*. If no match is found, then no ENTRY exists in the database for this RECORD, and the STRUCTURE for this RECORD is written to the database as an *insert*.

5.0 Testing Procedures

Three types of testing were run on the Postgresql RFC Archive System. The different scenarios were selected based on discussions that have been going on this past year

over table structure, pseudo-array (multiple values per row) versus single value per row and speculation over performance of Postgresql RDBMS inserts versus updates of a row.

Test 1 - For this test not much is running on the system, there are no other users on the system, oper's cron is shutdown and only shef_decode_raw is running.

Test 2 – For this test other users may or may not be on the system, oper's cron is running, one of the level 1 processors is being run by a user, shef_decode_pro is running and has been given a large number of files to process, and the shef_decode_raw application is running.

For Tests 1 and 2, four scenarios were given to shef_decode_raw application in order to collect information on inserts and updates to the two types of structures that make up the SHEF data value tables. These scenarios are:

- *inserts* to pseudo arrays tables
- *inserts* to single value per row tables
- *updates* to pseudo array tables
- *updates* to single value per row tables

Test 3 – For this test a very large queue of messages is to be processed by the shef_decode_raw application. (This could happen if a hardware problem stopped data processing and a large queue was created.) For the test not much else is running on the system, there are no other users on the system, oper's cron is shutdown and only shef_decode_raw is running.

For comparison purposes, tests 1 and 3 were also run on the Infomix system, ax1-nhdr.

The RFC Archive System application log_stats.tcl was used to collect performance information from the shefdecoder's daily log files. From this data, average posting time of records (or values) per second can be computed.

Example output of log_stats.tcl

Input file: /rfc_arc/logs/decoder/raw/logs/shef_decode_raw_log_1004

Number of Products = 2663
Number of records Processed = 264677
Avg Parse time = 0.00 seconds
Avg Post time = 3.83 seconds

Max Parse Time = 1 seconds
Max Post Time = 354 seconds

From the above information one can calculate the number of records posted per second; (Number of records processed divided by Number of products) divided by average post time. I.e.,

$$\frac{264677}{2663} = 99.39 \text{ records/products}$$

$$\frac{99.39}{3.83} = 25.95 \sim 26 \text{ records/second posted}$$

6.0 Test Results

6.1 Results of Tests 1 thru 3

Test 1 & 2, scenario 1: 10 messages with total of 50000 values all for the pecrsep table (pseudo-array), all new data

Informix		Postgresql			
Test 1		Test 1		Test 2	
posting values/sec	total run-time	posting values/sec	total run-time	posting values/sec	total run-time
158.7	6 min	32.0	27 min	26.5	32 min

Test 1 & 2, scenario 2: 14 messages with total of 52,931 values all for the pedrsep table (single value per row), all new data

Informix		Postgresql			
Test 1		Test 1		Test 2	
posting values/sec	total run-time	posting values/sec	total run-time	posting values/sec	total run-time
150.0	6 min	83.4	11 min	33.0	27 min

Test 1 & 2, scenario 3: 10 messages with total of 50000 values all for the pecrsep table (pseudo-array), all data previously posted

Informix		Postgresql			
Test 1		Test 1		Test 2	
posting values/sec	total run-time	posting values/sec	total run-time	posting values/sec	total run-time
162.3	6 min	30.0	28 min	23.3	36 min

Test 1 & 2, scenario 4: 14 messages with total of 52,931 values all for the pedrsep table (single value per row), all data previously posted

Informix		Postgresql			
Test 1		Test 1		Test 2	
posting values/sec	total run-time	posting values/sec	total run-time	posting values/sec	total run-time
152.5	6 min	67.0.	13 min	35.3	60 min

Test 3: 55,338 messages with total of 13,332,980 values. Messages are a variety of SHEF products such as RR1, RR2, RR3, RR7, RVF and RRS.

Informix		Postgresql	
posting values/sec	total run-time	posting values/sec	total run-time
142.5	7 hrs 16 min	45.5 value	23 hrs 20 min

(continues on next page)

In addition to these three tests, performance data for the Postgresql version of the shef_decode_raw application was collected from the ax2-nhdr daily log files for several days when the decoder was running with the live data-feed. During these periods there may or may not have been other users on the system. Typically there may be one or two programmers at OHD/HL, and one or two RAXUM support programmers, on ax2-nhdr at any given time. Oper's cron was running on these dates but it only had two tasks, run_vacuumdb and purge_files. The following table contains a summary of this information.

daily log file date	total number messages	total number values	average posting time (sec)	maximum posting time (sec)	posting values/sec
02/24/06	14571	546804	0.62	35	60.5
02/25/06	14610	523480	0.62	34	57.8
02/26/06	14661	530123	0.63	35	57.4
02/27/06	14706	529343	0.60	34	60
02/28/06	11140	430073	0.66	37	58.5
03/01/06	14921	563226	0.66	42	57.2
03/02/06	15076	561015	0.71	55	52.4
03/03/06	14972	563811	0.79	55	48.0
03/04/06	14886	578240	0.84	52	46.2
03/05/06	14729	568230	0.89	53	43.7
03/06/06	14129	536360	0.97	63	39.1
03/07/06	14712	605165	1.08	86	38.1
03/08/06	11153	437418	1.19	115	32.8
03/09/06	11036	831682	1.94	97	38.8

The declining posting rate (values/second) in the above table is due to the fact that the vacuumdb cron job, which had been running several times a day, began failing in early March. (This failure was due to a typo made when the script was being modified.) This decline in posting rate demonstrates the importance of ensuring the database is vacuumed routinely.

The “vacuum –z” cron job was corrected and testing with the vacuuming occurring twice per day was conducted. Results are presented in the table below. Average time to vacuum during this period was 70 minutes.

daily log file date	total number messages	total number values	average posting time (sec)	maximum posting time (sec)	posting values/sec
03/20/06	14884	569900	0.93	304	39
03/21/06	14921	588777	0.98	405	40.3
03/22/06	47443	479995	0.25	365	40.5
03/23/06	14602	557148	1.17	508	32.6
03/24/06	14888	561319	1.18	493	32
03/25/06	14829	561932	1.13	517	33.5
03/26/06	14685	561003	1.16	415	33
03/27/06	12368	460317	1.24	498	30
03/28/06	15088	560078	1.12	457	33.1
03/29/06	14722	561576	1.25	618	30.5

The question has been raised “If there is any benefit to doing a ‘vacuum full’ periodically?” to improve the performance of the raw shefdecoder. On 03/30/06 the shefdecoders were shutdown and oper’s cron was stopped while a “vacuum full” was performed on the test database, adb_ob7krf. It took 3 hours and 38 minutes to complete. All processes were restarted and for this test “vacuum –z” was set to run 6 times per day (average runtime of 60 minutes). Test results are shown below.

daily log file date	total number messages	total number values	average posting time (sec)	maximum posting time (sec)	posting values/sec
03/31/06	5540	336056	1.13	108	46.7
04/01/06	14758	562111	0.89	260	42.8
04/02/06	14365	533894	0.82	173	45.3
04/03/06	14748	546906	0.86	223	43.1
04/04/06	14631	566089	1.01	281	38.7
04/05/06	14715	545979	0.86	327	43.1
04/06/06	14432	544314	0.93	240	40.6
04/0726/06	14509	560815	0.96	305	40.3

The results indicate a slight improvement, but it is somewhat unclear whether there was actually any benefit from the “vacuum full”. The slight improvement could be due to going back to 6 times per day for the “vacuum –z”.

While beyond the scope of this study, tuning of the Postgresql engine appears to be the most likely way to significantly improve the speed of posting data to the archive database. The increased performance that tuning the Postgresql engine can provide has been examined for the IHFS DB in build ob6 at Northwest River Forecast Center; see the report “AWIPS Test Authorization Note – Test Results, POB6_OHD_A100768”.

6.2 Information on Deletes

In Informix when rows are deleted from a table with a simple sql query the user must be careful. If there are too many rows that fit the *where* clause, the query will fail with the error that the transaction was too long. Deletes in Postgresql are done much differently, i.e. the row is not actually removed but marked as deleted and becomes unviewable. The question was raised about how long it takes Postgresql to do deletes for a fairly large number of rows. For this test, all the rows older than “1982-01-01” in the pecrsep table (pseudo-array) were deleted and all rows older than “1982-01-01 00:00:00” in the pedrsep table (single value per row) were deleted. This test was performed only on the Postgresql RFC Archive DB. The results are in the following table.

Table	Table Type	Number of rows deleted	Time
pecrsep	pseudo-array	3,794	3 sec
pedrsep	single value per row	74,782	1 min 28 sec