BENCHMARKING POSTGRESQL FOR DATA WAREHOUSING

Eduardo Cunha de Almeida
Universidade Federal do Paraná
Centro Politécnico ZIP: 81531-990
Curitiba – PR – Brasil

Marcos Sfair Sunye
Universidade Federal do Paraná
Centro Politécnico ZIP: 81531-990
Curitiba – PR – Brasil

ABSTRACT
Often in a competitive market, corporations need tools to increase the accuracy and efficiency of the decisions. Facing this, several companies that have no resources to buy the commercial systems available because of the high costs, have no chance to buy a Decision Support Systems (DSS) platform. This paper is dedicated to a benchmark of PostgreSQL DBMS for data warehouse as part of a low cost platform. To accomplish this task we followed the TPC-H and DBT3 benchmarks to simulate a data warehouse workload. These benchmarks simulate multi-user environment and run complex queries, which execute: aggregations, nested sub queries, multi joins and others. Considering the results, we were able to point some PostgreSQL’s problems in the TPC-H execution, which were the main reason to execute the DBT3. Finally we demonstrate that this DBMS will become a real alternative to data warehousing with improvements on the optimizer and some structures.

KEYWORDS
Data Warehouse, Performance, Benchmark, Open source, PostgreSQL, DBMS.

1. INTRODUCTION
Today the market is demanding accurate and dynamic information to support its decisions, and the use of systems to support decision-making have increased and appeared to be a relevant competitive differential.

This demand has been supported by the utilization of Decision Support Systems (DSS) like On-line Analytical Processing (OLAP) and Data Mining tools over a Data Warehouse (DW).

Environments like DW are extremely expensive because of its computational size and strategic importance, and the high price is the main reason which universities and low capital companies do not have adopted.

Seems promising, therefore, a feasibility study of a low cost data warehouse platform intending to evaluate its performance.

In this study we will use software known as “open source”, like the operational system Linux and the DBMS PostgreSQL, and low cost hardware intending to obtain a feasible performance to stimulate the development and improvement of open source software focus on data warehousing.

So the objective of this paper is the achievement of a data warehouse feasibility study of PostgreSQL as a low cost platform.

Through this study we hope to demonstrate that future efforts in the development of the PostgreSQL DBMS are possible and promising in a data warehousing environment.
Using the open source model we hope to feed the interest in the development of another components of a data warehouse environment, like: On-Line Analytical Processing (OLAP) e Extract, Transform and Load (ETL).

Although this study focus a performance feasibility of a DBMS and a operational system with a low cost hardware, are also presented some briefs considerations about others data warehouse components that can influence in the final cost and performance results, as well as others quotations of studies made about DBMS.

In the performance measuring we will use the methodologies developed by the Open Source Development Lab (OSDL) and by the Transactional Processing Performance Council (TPC) and we will explain why it was necessary two different methodologies.

The paper is divided in four chapters, as follow: chapter two describes the data warehouse concept, the history and characteristics of PostgreSQL and some implementations which can increase the performance of PostgreSQL, chapter three describes the results and chapter four presents the conclusion.

2. POSTGRESQL AND DATA WAREHOUSE

2.1 Data Warehouse

Data Warehouse uses a methodology to integrate transactional databases into a new database. This new database stores all the company history and became a strategic tool to increase results and optimize investments of the companies.

Compared with transactional databases this kind of database has a different workload. DW just executes “SELECT” queries and the loading usually happens in certain periods (once a day, week or month).

The loading are usually big transactions because imports large amount of data from several sources. Some DW environments import millions of rows daily, like telecommunications companies.

2.2 PostgreSQL

PostgreSQL is the most advanced open-source database available anywhere. PostgreSQL came from Ingres DBMS [Drake, J.D. et al 2002].

The main characteristics of PostgreSQL are: referential integrity, lots of interfaces (ODBC, JDBC, PHP, and others), SQL92 e SQL99 specifications support, procedural languages support (Perl, Python, TCL and others), concurrence control avoiding read block when a write occurs and write control by writing in log before disk.

The optimizer uses statistics and several algorithms to execute queries. It has a genetic algorithm that makes different execution steps to the same query and then chose the best one. PostgreSQL query execution run as follow, according to [Lane, T. 2000], and can be visualized in the figure 1.

![Figure 1. PostgreSQL query execution](image-url)
Executions steps into the optimizer:
1 - The query is submitted to the parser that verifies the object’s definitions using the data dictionary;
2 - The query is rewrite;
3 - The planner builds an execution plan guided by the new query, the database statistics retrieved by the DBA and a genetic algorithm that generate several execution plan and chose the best one;
4 - The execution plan is executed.

In this work we use PostgreSQL 7.4.2. In the next version, 8, it will be implemented two features that can increase the DBMS performance. The features are:
- Multi-column index statistics. This achievement can dramatically increases some queries performance that we have run in this work, because large tables and complex queries use such index;
- Tablespaces that help the organization of large databases, distributing the object storage. This distribution can split data files and index files in different locations, decreasing the concurrency in I/O operations.

We suggest some implementations that can be done in PostgreSQL considering DW are:
- PAX (Partition Attribute Across) page strategy implementation purposed by [Ailamaki A.; et al, 1988];
- Encoded bitmap indexes purposed by [Wu, M. C. et al, 1998];
- Intra-query parallelism described by [Omiecinski, E. 1995].

These implementations can increase dramatically the performance of PostgreSQL based on the paper’s results.
- PAX groups each attribute into mini blocks. Query’s performance increases 11% to 42% with this kind of page strategy.
- The other improvement is a kind of grouping index called bitmap index. Bitmap index increases grouping queries and become a good choice to be implemented into low cardinalities attributes. If we consider just the encoded bitmap index implementation we can see the good performance results that Sybase IQ and MS SQL Server show in the TPC-H benchmark. Some papers and vendors recommend that bitmap index can be used into attributes with 0.1% of cardinality, which is 0.1% of all existent rows. Some DBMS like Sybase IQ implements specific grouping index depending on attribute cardinalities not just to the recommendation above.
- Considering the PostgreSQL’s open source characteristic we can think about the quality, low cost and freedom to change the source code as needed. Quality because the open source community is always concern about bug corrections in commitment with the rest of the community and low cost characteristic because the values of DBMS in Data Warehouse projects are almost 50% of the values of the entire project.

3. RESULTS

The benchmark execution begins with TPC-H that is the basis of this paper, after TPC-H we executed DBT3.

First we executed TPC-H 100 GB scale then the 1GB scale and finally the DBT3 1GB because of the results.

The environment configuration is: OSDL DBT3 version 1.4 and TPC-H version 2.0.0; Scale factor 1GB and 100GB; Loading using flat files.
Softwares used are: PostgreSQL 7.4.2; Mandrake Linux 64bits (kernel 2.6.5); Java SDK 1.4.2_04; PostgreSQL JDBC pg74.1jdbc3.jar; TPC-H utilities (DBGEN and QGEN)
Hardware used is: Dual Opteron 64bits Model 240 1.4GHz; 4 GB RAM; 960 GB Disk RAID 0.
The programs to load the database and the execution of the TPC-H benchmark were written in JAVA and shell script. The DBT3 benchmark was executed by the package provide by OSDL.
JavaSDK 32-bits version was used because Sun Microsystems does not provide an AMD Opteron stable 64-bits version until the beginning of this work and became a reason to compile the Linux kernel with 32-bits support.
We load the database three times using kernel the versions 2.4.24 and 2.6.5 to measure the differences and retrieve maximum performance from the system. First we created a monolithic script that is a script without load balance.

Analyzing these times we decided to use the kernel version 2.6.5 and split the script to use both processors.

The load balance is divided in two. The first part build LINEITEM table and the second part build the rest of the database.

We achieved an improvement of 32 % using a 64-bits OS and the kernel version 2.6.x compared to a 32-bits OS and kernel 2.4.x in the loading phase.

Compared to the Sybase IQ with a similar environment we have a 70% less performance in the loading phase, but 86.16 % less cost.

The Sybase IQ environment described into TPC website costs US$45.021 and takes 6:16:00 hours to load the 100 GB database on a similar machine.

The Table 1 shows the loading results and the figure 2 shows the TPC-H results.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Kernel</th>
<th>Scale(GB)</th>
<th>Script</th>
<th>Data load</th>
<th>PK and index</th>
<th>Total load time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debian 32bits</td>
<td>2.4.24</td>
<td>100</td>
<td>Monolithic</td>
<td>11:37:59</td>
<td>12:22:04</td>
<td>24:00:04</td>
</tr>
<tr>
<td>Mandrake 64bits</td>
<td>2.6.5</td>
<td>100</td>
<td>Monolithic</td>
<td>07:24:23</td>
<td>13:42:31</td>
<td>21:06:54</td>
</tr>
<tr>
<td>Mandrake 64bits</td>
<td>2.6.5</td>
<td>100</td>
<td>Distributed</td>
<td>06:14:30</td>
<td>09:58:31</td>
<td>16:13:01</td>
</tr>
<tr>
<td>Mandrake 64bits</td>
<td>2.6.5</td>
<td>1</td>
<td>Distributed</td>
<td>00:03:40</td>
<td>00:03:39</td>
<td>00:07:19</td>
</tr>
</tbody>
</table>

Table 1. Loading results

Figure 2. The power test result

Some queries run near or faster than Sybase and MS SQL Server like queries number 11 and 18. These queries run complex operations like sub-queries and sub-queries inside the HAVING clause.

The queries that take the longest time shows PostgreSQL problems compared to another DBMS. These queries number are 4, 8, 9, 10, 19, 20 and 22. The operations executed by these queries are:

- Sub-queries inside other sub-queries;
- EXISTS and NOT EXISTS operator;
- Aggregation with in-line view, that is sub-query inside the FROM clause;
- Selection by date.

Most of these queries make the selection by date, which is a data warehouse regular use and DBMS like Sybase has specific type of indexes for this purpose.

Another bottleneck that we have found is the optimizer.
The optimizer creates an execution plan after the query rewrite based on the database statistics. Here we can point two important factors: a periodic update of the statistics and the query rewrite.

The first factor is the statistic update that we have made after the database load. This is enough to the execution of the benchmark.

The second factor is the query rewrite and this process is executed by the DBMS so its not possible to change without verify the source code. In this case is better to change the SQL text before submit to the system and TPC-H does not allow this.

We use the original queries in the beginning of the tests and some was abort by timeout, because PostgreSQL does not generate a good execution plan. These problems just allow the execution of power test.

Then we verify that the bad execution plans was generate because of the query rewrite, pointed as the major update to the next PostgreSQL’s version.

We can analyze in the query number 19 that the “join” operations and some selections are write inside each “OR” operation. Into the rewritten query these common segments is left outside the “OR” operations. The optimizer without human assistance must do this kind of rewrite process. The new query was developed by OSDL for DBT3 benchmark and the texts are described in figure 3.

We decide to put an interruption timeout clause because of the bad plans generation. The time is 25000 seconds.

Analyzing the plans and the queries with the timeout parameter we were able to verify the problems of the DBMS.

After we interrupt the queries by timeout, some of them were executed without this parameter. Even without timeout the query number 19 must be interrupted because exceeded 72 hours of execution in the 100 GB scale.

In the 1 GB scale the same query was interrupted after 24 hours of execution without retrieving something. When the rewrite of this query was made the execution takes an average of 25,64 seconds confirming the PostgreSQL optimizer problem more specifically in the query rewrite module.

The execution plans are described in figure 4.

```
Query 19 – TPC-H version

Select sum(l_extendedprice* (1 - l_discount)) as revenue
from
lineitem,
part
where

\{ 
  p_partkey = l_partkey
  and p_brand = [BRAND1]
  and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
  and l_quantity >= [QUANTITY1] and l_quantity <= [QUANTITY1] + 10
  and p_size between 1 and 5
  and l_shipmode in ('AIR', 'AIR REG')
  and l_shipinstruct = 'DELIVER IN PERSON'
\}

or

\{ 
  p_partkey = l_partkey
  and p_brand = [BRAND2]
  and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
  and l_quantity >= [QUANTITY2] and l_quantity <= [QUANTITY2] + 10
  and p_size between 1 and 10
  and l_shipmode in ('AIR', 'AIR REG')
  and l_shipinstruct = 'DELIVER IN PERSON'
\}

...
```

```
Query 19 – OSDL version

Select sum(l_extendedprice* (1 - l_discount)) as revenue
from
lineitem,
part
where

\{ 
  p_partkey = l_partkey
  and p_brand = [BRAND1]
  and l_quantity >= [QUANTITY1] and l_quantity <= [QUANTITY1] + 10
  and l_shipmode in ('AIR', 'AIR REG')
  and l_shipinstruct = 'DELIVER IN PERSON'
\}

or

\{ 
  p_partkey = l_partkey
  and p_brand = [BRAND2]
  and l_quantity >= [QUANTITY2] and l_quantity <= [QUANTITY2] + 10
  and l_shipmode in ('AIR', 'AIR REG')
  and l_shipinstruct = 'DELIVER IN PERSON'
\}

...
```

Figure 3. Query 19 texts
Just to mention Sybase executes this query in the 100 GB scale in 971 seconds. It is not possible to compare the test results with TPC-H because the tests were stopped by timeout or interrupted with some days of execution; in the last case it is unacceptable.

Considering the situation above we decide to run the DBT3 benchmark, which considers modification in the queries. In this condition PostgreSQL executed the workload.

The measures generated by DBT3 are:
- Power@size = 332,35
- Throughput@size = 224,85
- Composite = 273,37

This result just can be compared to other OSDL result and is not conclusive to compare to another DBMS that do not execute the OSDL benchmark.

<table>
<thead>
<tr>
<th>Benchmark’s execution summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scale (GB)</strong></td>
</tr>
<tr>
<td>Status</td>
</tr>
<tr>
<td>Timeout per query (sec)</td>
</tr>
<tr>
<td>Exec. Time (sec)</td>
</tr>
<tr>
<td>Without Timeout Status</td>
</tr>
<tr>
<td>Exec. Time</td>
</tr>
</tbody>
</table>

(*) only power test
4. CONCLUSION

In this work we executed benchmarks that test the low cost platform performance using PostgreSQL and Linux, pointing the problems and verifying if even considering the problems the platform is feasible.

We also verify structures, which can increase the PostgreSQL’s performance.

We have shown that PostgreSQL DBMS version 7.4.2 is not able to execute satisfactory TPC-H benchmark the reason why we executed DBT3 benchmark. Our feasibility expectations of a low cost data warehouse with our environment were not fully reached.

It is important to mention the considerable increase of the performance from version 7.3.4 to 7.4.2 with the development of some aggregation features. Then we can expect, in a near future, to compared to another DBMS like Sybase and SQL Server. The differences can become shorter with the natural open source development and the implementation of the structures that we have suggested.

We expected that PostgreSQL becomes a real alternative to data warehouse project in any scale in the next version.

The fix of the optimizer fix is considered crucial even with the development of the structure suggested in this work and is the main improvement of the next version.

To use PostgreSQL in a data warehouse project with the actual version 7.4.2 some reservations must be considered:

- The organization does not have resources to buy a DBMS more mature;
- Accept the low performance because of the problems pointed in this work;
- Constantly monitors the queries submitted to the system to verify if a rewrite is needed. In this case if the number of users grows the cost of the manual rewrite grows too.

The use of Linux OS can be assured in the TPC benchmarks and its use can bring very good results as we can see in the TPC website.

As future work we can suggest the development into PostgreSQL of some structures that can increase the performance. These implementations are:

- PAX pages proposed by [Ailamaki A.; et al, 1988]. This strategy can decrease I/O operations as described by the author;
- Bitmap index proposed by [Wu, M. C. et al, 1998]. This kind of index can increase aggregation queries;
- Intra-query parallelism describe by [Omiecinski, E. 1995]. The parallelism can increase the performance of sub-queries and in-line views that we have low performance in the PostgreSQL.
- The use of clusters, because this work uses a multi-processor server. The use of a cluster is a good alternative when the processing power must be improved and has a low cost if compared to a multi-processor machine;
- Development of a similar methodology to TPC-H and DBT3 to take advantages of object oriented features of PostgreSQL.

Other works that can be developed using the open source model to the other data warehouse’s components are:

- OLAP tool;
- ETL tool using the works proposed by [Jung, I. Et al, 2000] and [Kavalco, G. 2001].

REFERENCES


Revision 2.0.0., 2003
