Development of a Parallel DBMS on the Basis of PostgreSQL

C. S. Pan
kvapen@gmail.com
South Ural State University

Abstract. The paper describes the architecture and the design of PargreSQL parallel database management system (DBMS) for distributed memory multiprocessors. PargreSQL is based upon PostgreSQL open-source DBMS and exploits partitioned parallelism.

Keywords: partitioned parallelism; postgresql; parallel dbms.

1. Introduction

Currently open-source PostgreSQL DBMS [1] is a reliable alternative for commercial DBMSes. There are many both practical database applications based upon PostgreSQL and research projects devoted to extension and improvement of PostgreSQL.

One of the directions mentioned above is to adapt PostgreSQL for parallel query processing. In this paper we describe the architecture and design of PargreSQL parallel DBMS for analytical data processing on distributed multiprocessors. PargreSQL represents PostgreSQL with embedded partitioned parallelism.

The paper is organized as follows. Section 2 briefly discusses related work. Section 3 gives a description of the PostgreSQL DBMS architecture. Section 4 introduces design principles and architecture of PargreSQL DBMS. The results of experiments on the current partial implementation are shown in section 5. Section 6 contains concluding remarks and directions for future work.

2. Related Work

The research on extension and improvement of PostgreSQL DBMS includes the following.

In [2] native XML type support in PostgreSQL is discussed. Adding data types to provide support of HL7 medical information exchange standard in PostgreSQL is described in [3]. The authors of [4] propose an image-handling extension to

PostgreSQL. In [5] an approach to integration of PostgreSQL with the Semantic Web is presented.

There are papers investigating adoption of PostgreSQL for parallel query processing as well. In [6] the authors introduce their work on extending PostgreSQL to support distributed query processing. Several limitations in PostgreSQL's query engine and corresponding query execution techniques to improve performance of distributed query processing are presented. ParGRES [7] is an open-source database cluster middleware for high performance OLAP query processing. ParGRES exploits intra-query parallelism on PC clusters and uses adaptive virtual partitioning of the database. GParGRES [8] exploits database replication and inter- and intra-query parallelism to efficiently support OLAP queries in a grid. The approach has two levels of query splitting: grid-level splitting, implemented by GParGRES, and node-level splitting, implemented by ParGRES.

In [9] building a hybrid between MapReduce and parallel database is explored. The authors have created a prototype named HadoopDB on the basis of Hadoop and PostgreSQL, that is as efficient as a parallel DBMS, but as scalable, fault tolerant and flexible as MapReduce systems. PostgreSQL is used as the database layer and Hadoop as the communication layer.

Out contribution is embedding partitioned parallelism [10] into PostgreSQL. We use methods for parallel query processing, proposed in [11] and [12].

3. PostgreSQL Architecture

PostgreSQL is based on the client-server model. A session involves three processes into interaction: a frontend, a backend and a daemon (see fig. 1).

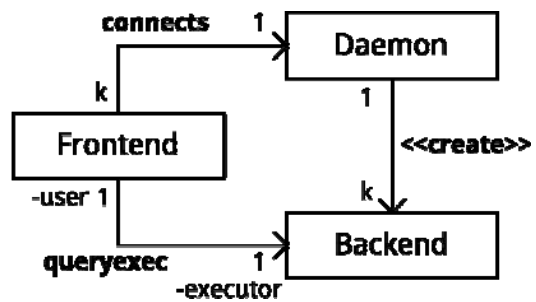

Fig. 1. PostgreSQL processes

The daemon handles incoming connections from frontends and creates a backend for each one. Each backend executes queries received from the related frontend. The activity diagram of a PostgreSQL session is shown in fig. 2.

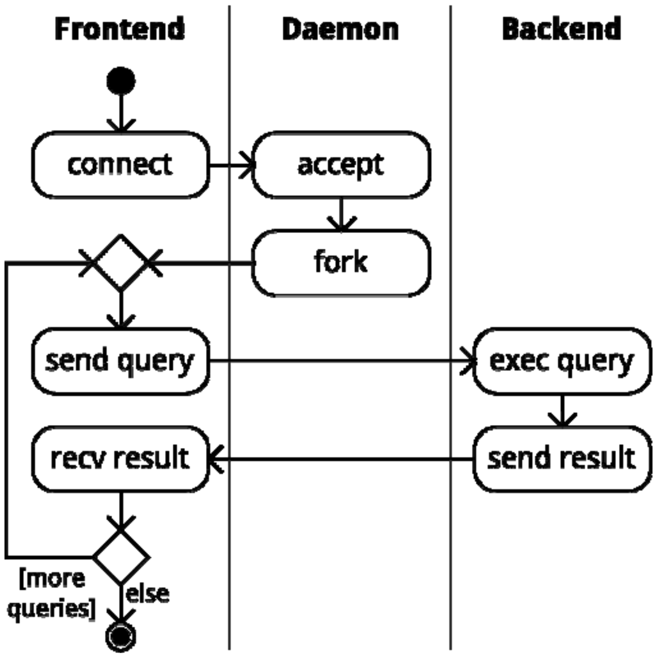

Fig. 2. A PostgreSQL session

There are following steps of query processing in PostgreSQL: *parse*, *rewrite*, *plan/optimize*, and *execute*.

Respective PostgreSQL subsystems are depicted in fig. 3. *Parser* checks the syntax of the query string and builds a parse tree. *Rewriter* processes the tree according to the rules specified by the user (e.g. view definitions). *Planner* creates an optimal execution plan for this query tree. *Executor* takes the execution plan and processes it recursively from the root. *Storage* provides functions to store and retrieve tuples and metadata.

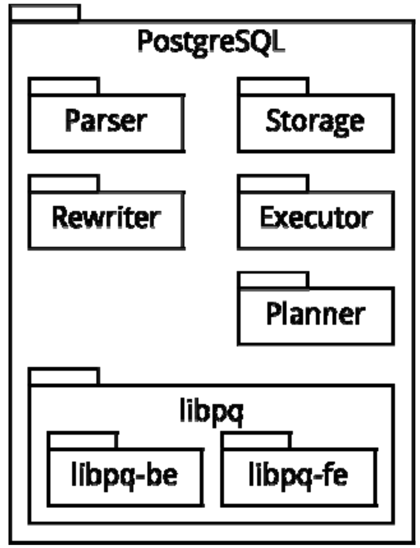

Fig. 3. PostgreSQL subsystems

libpq implements frontend-backend interaction protocol and consists of two parts: the frontend (*libpq-fe*) and the backend (*libpq-be*). The former is deployed on the client side and serves as an API for the end-user application. The latter is deployed on the server side and serves as an API for libpq-fe, as shown in fig. 4.

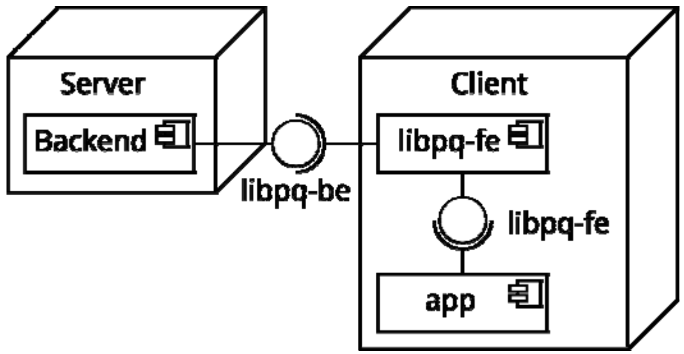

Fig 4. PostgreSQL deployment

4. PargreSQL Architecture

PargreSQL utilizes the idea of partitioned parallelism [12] as shown in fig. 5. This form of parallelism supposes partitioning relations among the disks of the multiprocessor system.

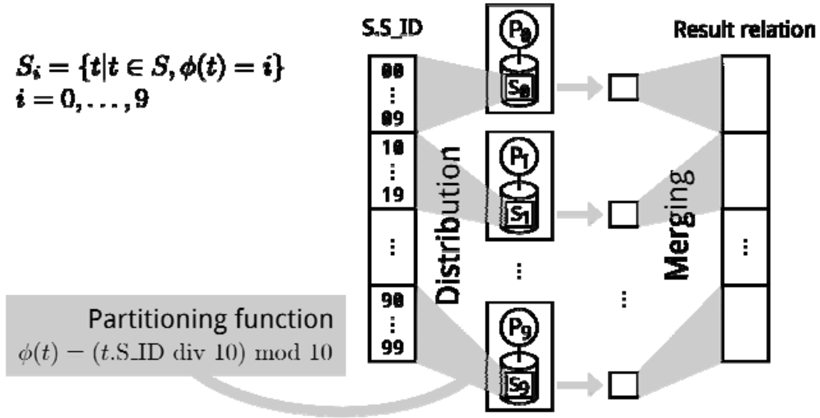

Fig. 5. Parallel query processing

The way the partitioning is done is defined by a *fragmentation function*, which for each tuple of the relation calculates the number of the processor node which this tuple should be placed at. A query is executed in parallel on all processor nodes as a set of parallel *agents*. Each agent processes its own fragment and generates a partial query result. The partial results are merged into the resulting relation.

The architecture of PargreSQL, in contrast with PostgreSQL, assumes that a client connects to two or more servers (see fig. 6).

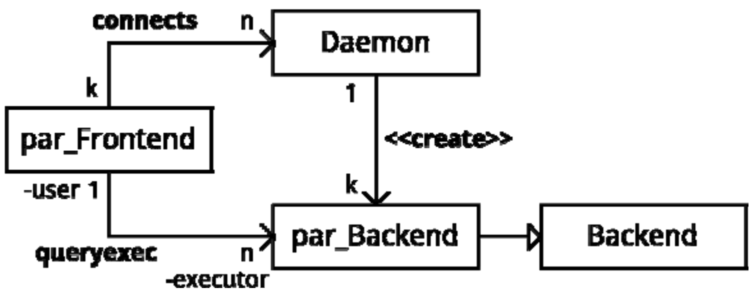

Fig. 6. PargreSQL processes

The interaction sequence is shown in fig. 7. As opposed to PostgreSQL there are many daemons running in PargreSQL. A frontend connects to each of them, sends the same query to many backends, and receives the result relation.

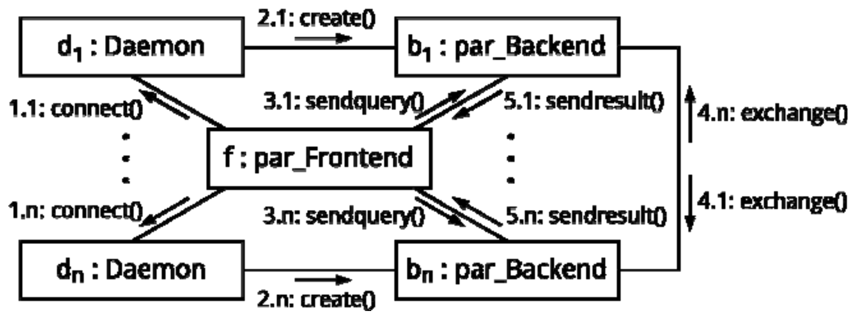

Fig. 7. Interaction of PargreSQL clients and servers

Parallel query processing in PargreSQL is done in more steps: *parse*, *rewrite*, *plan/optimize*, ***parallelize***, *execute*, and ***balance***. During the query execution each agent processes its own part of the relation independently so, to obtain the correct result, transfers of tuples are required. *Parallelization* stages creation of a parallel plan by inserting special exchange operators into the corresponding places of the plan. *Balance* provides load-balancing of the server nodes.

PargreSQL subsystems are depicted in fig. 8. PostgreSQL is one of them. PargreSQL development involves changes in Storage, Executor, and Planner subsystems of PostgreSQL.

The changes in the old code are needed in order to integrate it with the new subsystems. *par_Storage* is responsible for storing partitioning metadata of the relations. *par_Exchange* encapsulates the *exchange* operator implementation. *Exchange* operator is meant to compute the exchange function ψ for each tuple of the relation, send “alien” tuples to the other nodes, and receive “own” tuples in response.

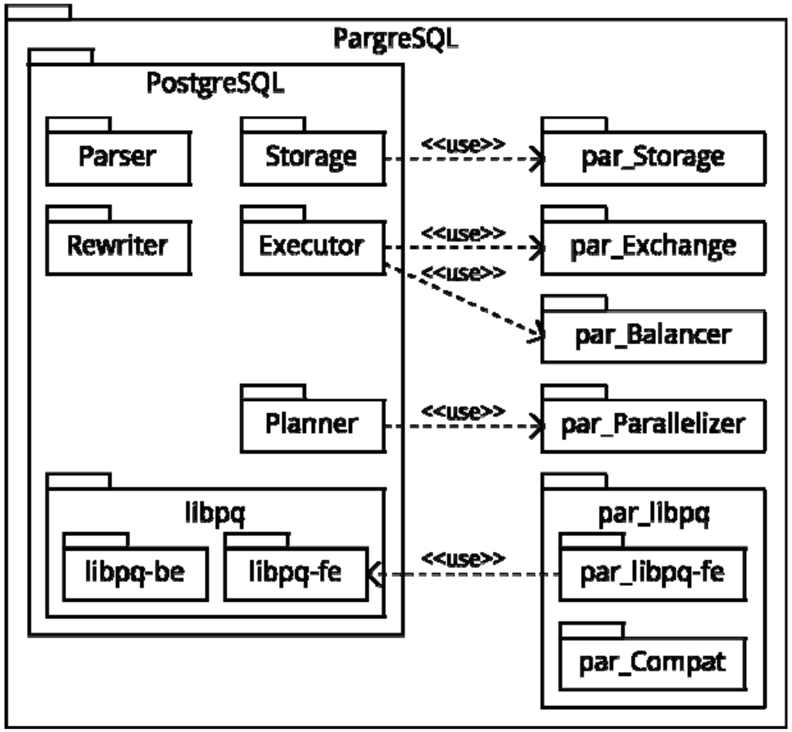

Fig. 8. PargreSQL subsystems

There are however some new subsystems which do not require any changes in the old code: *par_libpq-fe* and *par_Compat*. *par_libpq-fe* is a wrapper around *libpq-fe*, it is needed in order to propagate queries from an application to many servers. *par_Compat* makes this propagation transparent to the application.

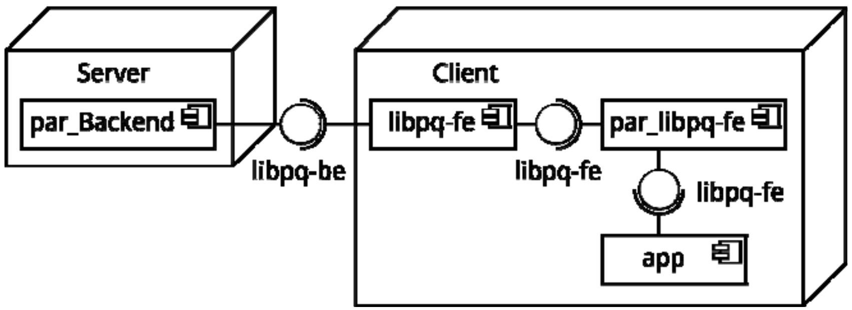

Fig. 9. PargreSQL deployment

The only difference of the deployment schemes (see fig. 9) is that there is one more component on the client side – the *libpq-fe* wrapper.

4.1. par_libpq Design

par_libpq subsystem consists of par_lib-fe library and a set of macros (par_Compat).

par_libpq-fe is a library that is linked into frontend applications instead of the original PostgreSQL libpq-fe, arouch which it is a wrapper. Its design is illustrated with a class diagram in fig. 10.

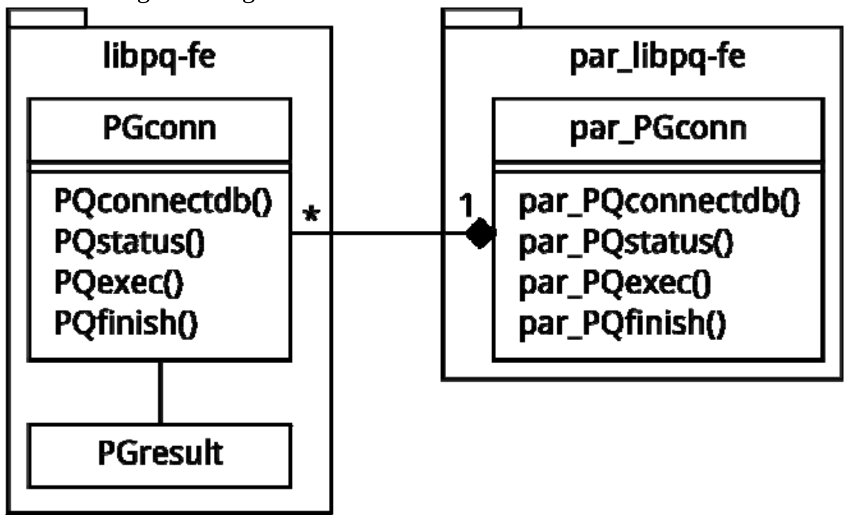

Fig. 10. PargreSQL libpq-fe wrapper

The idea is to use the original library for connecting to many servers simultaneously.

par_Compat is a set of C preprocessor definitions for transparent usage of *par_libpq-fe*. An example of what these macros are is given in fig. 11.

```
#define PGconn par_PGconn
#define PQconnectdb(X) par_PQconnectdb()
#define PQfinish(X) par_PQfinish(X)
#define PQstatus(X) par_PQstatus(X)
#define PQexec(X,Y) par_PQexec(X,Y)
```

Fig. 11. PargreSQL compatibility macros

Using these macros an application programmer can switch from PostgreSQL to PargreSQL without global changes in the application code.

4.2. Exchange Operator Design

Exchange operator [11, 12] serves to exchange tuples between the parallel agents. It is inserted into execution plans by Parallelizer subsystem. The operator's architecture is presented in fig. 12.

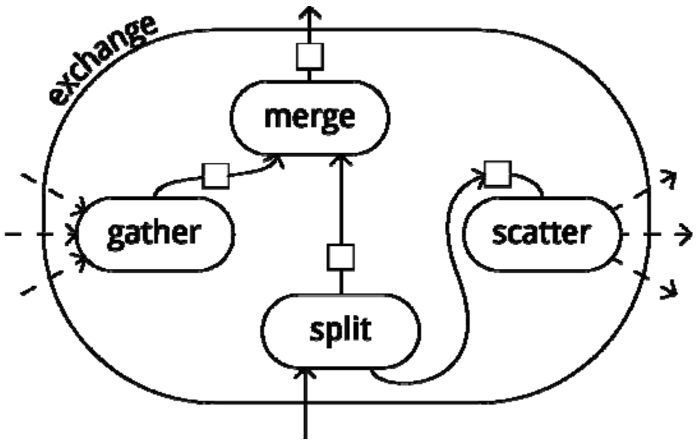

Fig. 12. Exchange operator architecture

Fig. 13 shows new classes (grouped in par_Exchange package) that implement *exchange* operator.

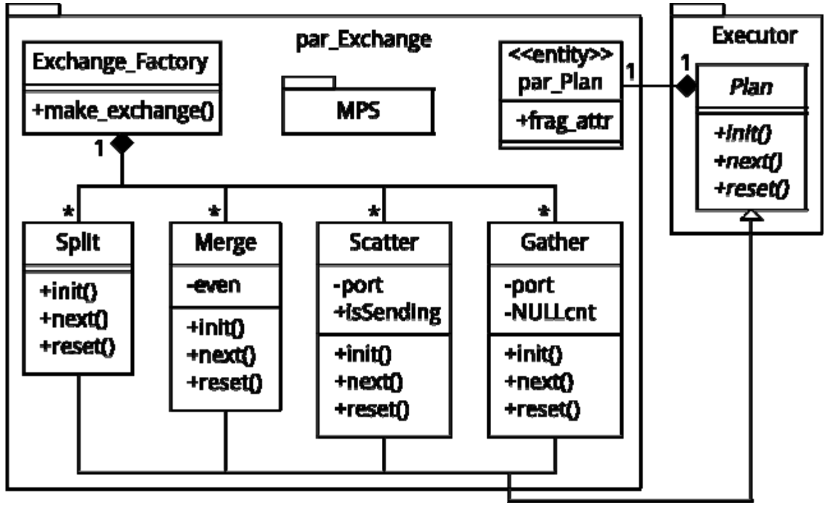

Fig. 13. Exchange operator classes

MPS subsystem (Message Passing System) is used by Scatter and Gather to transmit tuples. Its interface is like MPI reduced to three methods: ISend, IRecv, and Test. They are actually implemented on top of MPI.

Figs. 14, 15, 16, and 17 show algorithms for **next()** method of four exchange subnodes.

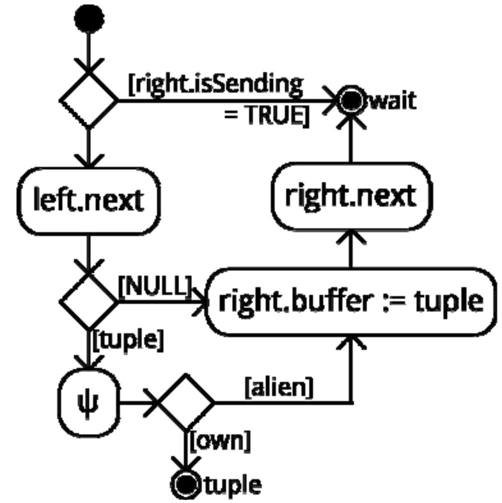

Fig. 14. **Split.next()** method

Split is meant to calculate the exchange function for each tuple and to choose whether to keep the tuple on the processor node or send it to other processor node.

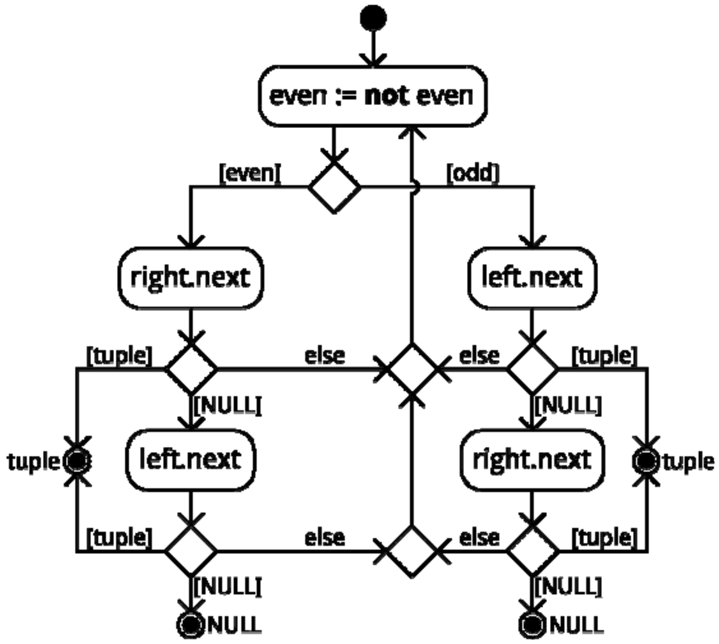

Fig. 15. **Merge.next()** method

Merge merges tuples from *Gather* and *Split*.

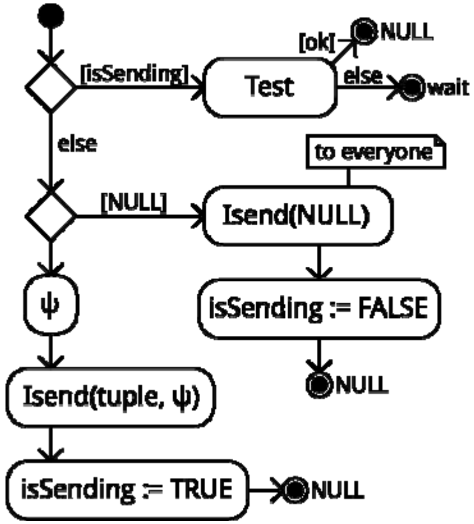

Fig. 16. **Scatter.next()** method

Scatter sends tuples coming from *Split* to other processor nodes.

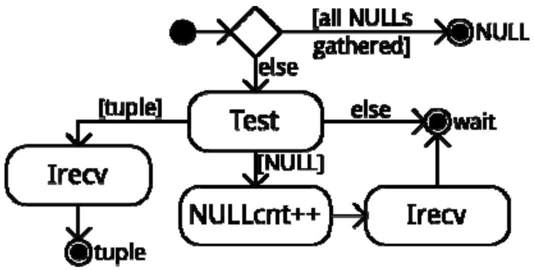

Fig. 17. **Gather.next()** method

Gather does the opposite, receiving tuples from other processor nodes.

5. Experimental Evaluation

At the moment we have implemented *par_libpq* and *par_Exchange* subsystems of PargreSQL. The implementation has been tested on the following query:

```
select * from tab where tab.col % 10000 = 0
```

The query has been run against table *tab* consisting of 10^8 tuples. The speedup over PostgreSQL is shown in fig. 18.

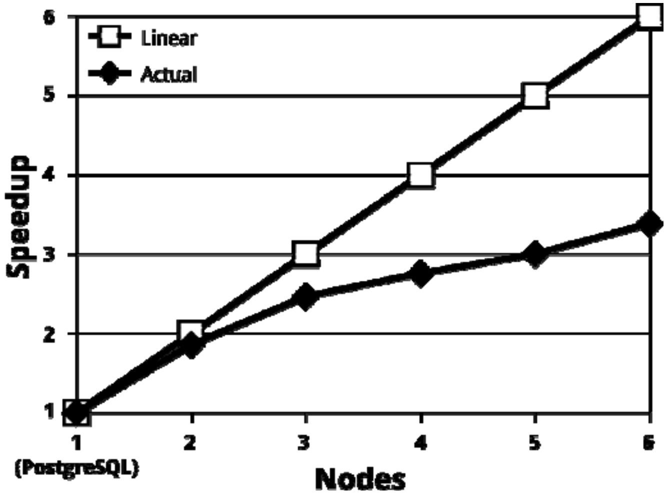

Fig. 18. PargreSQL speedup

6. Conclusion

In this paper we have described the architecture and the design of PargreSQL parallel DBNS for distributed memory multiprocessors. PargreSQL is based upon PostgreSQL open-source DBMS and exploits partitioned parallelism.

There are following issues in out future research. We plan to complete the implementation and to investigate its speedup and scalability. The future research is also going to be concentrated on implementing data updates, transactions and fault tolerance.

References

[1] M. Stonebraker, G. Kemnitz. The POSTGRES next generation database management system. Commun ACM, 34:78—92. October 1991.

[2] N. Samokhvalov. XML Support in PostgreSQL. In Sergei D. Kuznetsov, Andrey Fomichev, Boris Novikov, and Dmitry Shaporenkov, editors, SYRCoDIS, volume 256 of CEUR Workshop Proceedings. CEUR-WS.org, 2007.

[3] Y. Havinga, W. Dijkstra, A. de Keijzer. Adding HL7 version 3 data types to PostgreSQL. CoRR, abs/1003.3370, 2010.

[4] D. Guliato, E. V. de Melo, R M. Rangayyan, R. C. Soares. POSTGRESQL-IE: An Image-handling Extension for PostgreSQL. J. Digital Imaging, 22(2):149—165, 2009.

[5] D. V. Levshin, A. S. Markov. Algorith,s for integrating PostgreSQL with the semantic web. Programming and Computer Software, 35(3):136—144, 2009.

[6] R. Lee, M. Zhou. Extending PostgreSQL to Support Distributed/Heterogeneous Query Processing. In Kotagiri Ramamohanarao, P. Radha Krishna, Mukesh K. Mohania, and Ekawit Nantajeewarawat, editors, DASFAA, volume 4443 of Lecture Notes in Computer Science, pages 1086—1097. Springer, 2007.

[7] M. Paes, A. A. B. Lima, P. Valduriez, M. Mattoso. High-Performance Query Processing of a Real-World OLAP Database woth ParGRES. In Jose M. Laginha M. Pal ma, Patrick Amestoy, Michel K. Dayde, Marta Mattoso, and Joao Correia Lopes, editors, VECPAR, volume 5336 of Lecture Notes in Computer Science, pages 188—200. Springer, 2008.

[8] N. Kotowski, A. A. B. Lima, E. Pacitti, P. Valduriez, M. Mattoso. Parallel query processing for OLAP in grids. Concurrency and Computation: Practice and Experience, 20(17):2039—2048, 2008.

[9] A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Rasin, A. Silberschatz. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. PVLDB, 2(1):922—933, 2009.

[10] D. J. DeWitt, J. Gray. Parallel Database Systems: The Future of High Performance Database Systems. Commun. ACM, 35(6):85—98, 1992.

[11] L. B. Sokolinsky. Organization of Parallel Query Processing in Multiprocessor Database Machines with Hierarchical Architecture. Programming and Computer Software, 27(6):297—308, 2001.

[12] A. V. Lepikhov, L. B. Sokolinsky. Query Processing in a DBMS for cluster systems. Programming and Computer Software, 36(4):205—215, 2010.