

Replication in Node Partitioned Data Warehouses

Pedro Furtado

University of Coimbra
Departamento de Engenharia Informática
Pólo II - Pinhal de Marrocos
3030 - 290 Coimbra
Portugal
pnf@dei.uc.pt

Abstract

In this paper we concentrate on guaranteeing efficient availability and promoting manageability in a node-partitioned data warehouse (NPDW). The objective is that the system be always-on and always efficient even when entire parts of it are taken offline for maintenance and management functions such as loading with new data or other DBA functionality. Replication has already been studied for parallel databases in general. We investigate how alternative replication strategies can be applied to the NPDW context and analyze advantages and drawbacks against metrics.

1. Introduction

Parallel architectures can speedup significantly the processing over large data warehouses. We have been pursuing the idea of replacing fully-dedicated and powerful servers by a possibly non-dedicated network of low-cost, under-utilized computers to hold and process data warehouses. The data warehouse can reach giga or even terabytes and is typically organized as a set of multidimensional schemas [10]. There are typically some very big relations – facts - storing historical detail, such as each individual sale of each product in each store of a retail chain, and smaller relations – dimensions – with descriptive properties for the dimensions (e.g. product, store, time). In that context, partitioning refers to dividing relations into nodes somehow, to take advantage of parallel node processing. We have discussed horizontal partitioning strategies for NPDW in [5] and showed that a careful partitioning strategy over a switched network environment can achieve acceptable speedups. However,

availability is an issue in such a context, so that availability-oriented replication becomes a major necessity as a way to provide availability. A replica is a “standby” copy of some data that can be activated at any moment in case of unavailability or failure of the node holding the “original”, so that processing resumes as usual. If processing with unavailable nodes is implemented efficiently, unavailability becomes less onerous to the whole system and it also becomes feasible to stop a set of nodes for data loading, maintenance, upgrading or other management activities, without any major repercussions to processing. The system remains always on and always efficient.

Replica placement has been studied in the context of generic parallel and distributed databases in which the relations are not partitioned [2, 7, 8, 9, 15, 16]. We review those works in the related work section. In this paper we discuss replication for availability in the NPDW context and discuss their use for both tolerating node failures and allowing multiple nodes to be offline simultaneously for loading or administration. We compare the approaches from the perspective of efficiency. Our main contributions include: showing how replication strategies can be applied to a workload-based pre-partitioned NPDW setting and how processing can incorporate the replicas in case of node failures; analyzing alternatives against relevant metrics; evaluating the alternatives with emphasis on efficiency and flexibility for allowing multiple offline nodes; analyzing the tradeoff between efficiency and the capacity to take multiple nodes offline simultaneously. The paper is organized as follows: section 2 discusses related work. Section 3 overviews the Node Partitioned Data Warehouse. Sections 4 and 5 discuss replication alternatives and section 6 compares the approaches. Section 7 contains concluding remarks and future work.

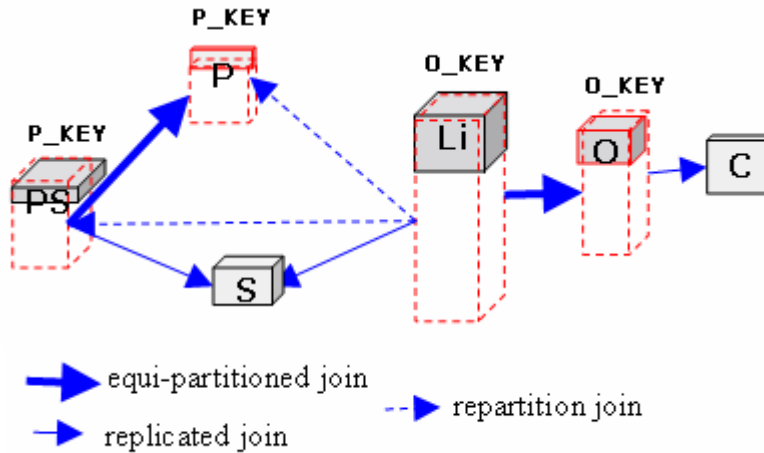


Figure 1: Basic Partitioning Example in NPDW (TPC-H schema)

2. Related Work

The most relevant related work for this paper concerns on replication strategies, but we also review briefly partitioning. Some of the most promising partitioning and placement approaches focus on query workload-based partitioning choice [14, 19]. The idea in those works is to use the query workload to determine the most appropriate partitioning attributes, which should be related to typical query access patterns. All those works focus mainly on hash-partitioning, for efficient parallel join processing [3, 11], also reviewed in [18]. Our previous work on the NPDW [4, 5] proposes and analyzes generic data partitioning strategies independently of the underlying database server and targeted at node partitioned data warehouses. Our purpose in this paper is to study availability and replication concerns to the NPDW design.

Replication has been studied in the parallel database context. In Tandem’s NonStop SQL [15] the use of mirrored disk drives offers a high level of availability but does a poor job of distributing the load of a failed processor. If a processor fails, the substitute processor will have to handle the disks of the failed processor as well as its own, essentially doubling the processing time. In this paper we apply this strategy as the Full replication (FR) option. Teradata’s scheme [16] assumes relation clusters (group of nodes) and can backup a partitioned copy of a relation by placing it in the N-1 other nodes of the relation cluster with N nodes. Although this scheme balances the processing in case of failures, if more than one node is unavailable in the cluster the system stops. In chained declustering [7, 8, 9] two declustered copies are kept such that the

fragments of the second declustered copy are placed in different nodes from the ones of the primary copy. This strategy improves availability while maintaining the performance level of the Teradata scheme. In [16] interleaved declustering divides the disks into clusters and fully declusters relation partitions into the corresponding cluster. In [2] the authors compare high-availability media recovery techniques in a generic OLTP environment, including Teradata’s interleaved declustering.

Recent work on replication includes [1, 13]. The authors use data replication to improve data availability and query load balancing while dealing with consistency problems. They propose a lazy preventive data replication solution in [13] and a strategy to scale-up the solution in [1]. The work in [12] studies similar approaches when applied to WAN environments. They identify the most crucial bottlenecks of the existing protocols, and propose optimizations that alleviate the identified problems.

While these works focus on generic replication strategies for availability considering non-partitioned relations and/or OLTP loads, we discuss, analyze and evaluate replication strategies on the specific context of the Node-Partitioned Data Warehouse and also consider if the strategies allow multiple nodes to be offline simultaneously for maintenance or management.

3. The NPDW

The NPDW is a design for efficient processing of the data warehouse over low-cost computer nodes on a possibly non-dedicated, switched network. The objective is not to assume any specialized hardware or interconnects, so that the NPDW is able to run for instance in a 100Mbps switched LAN. Parallelism is

obtained by dividing the data set initially into the disks of individual nodes, so that each node is able to access its data locally and data is exchanged between nodes when necessary. In order to deliver a near to linear speedup over the node-partitioned NPDW context, it is necessary to find suitable partitioning and placement strategies for the data, which may reduce the need to exchange data between nodes. This issue is discussed in detail in [4, 5] and we only review it in this paper. Our objective was for the NPDW to be able to process efficiently not only simple star schemas [10], but also more complex data warehouse schemas such as TPC-H [17]. In a partitioning and placement scheme, each relation can essentially be partitioned (divided into partitions or fragments) or copied in their entirety into all nodes of a group. In order to simplify our discussion, we assume that they are either copied or partitioned into all nodes, that is, the group is “all nodes”. We also simplify the discussion by considering homogeneous nodes, so that each node has the same load. This constraint can be eliminated by taking into account node performances in the initial placement and subsequent reorganizations for load balancing. Relations that are copied into all nodes are also denoted as replicated relations. The decision to replicate relations for performance reasons is an output from partitioning, not availability-related replication, but the resulting replicas are, of course, also useful for availability. In order to distinguish a replica dictated by a partitioning algorithm from one dictated from availability we denote partitioning replication as P-replication.

Partitioned relations can be divided using a round-robin, random, range or hash-based scheme. The NPDW uses horizontal hash-partitioning, as this approach facilitates key-based tuple location and join operations. Figure 1 shows the partitioning and placement of relations for the TPC-H benchmark [17] after the workload-based algorithm in [5] was applied (LI-lineitem, O-orders, PS-partsupp, P-part, S-supplier, C-customer). In that Figure, dashed rectangles represent fully-partitioned relations; dashed arrows represent “repartition joins” (RJ- joins that require data to be shipped between nodes); bold arrows represent “equi-partitioned joins” (EJ- joins that do not require data to be shipped between nodes because the intervening data sets are partitioned by the join key) and normal arrows represent “P-replicated joins” (RRJ – joins that do not require data to be shipped between nodes because one of the intervening relations is P-replicated). Repartitioning refers to the need to exchange data between nodes in order to reorganize two data sets so that they become equi-partitioned (partitioned by the same attribute). In order to choose the most appropriate partitioning alternative, we must use a strategy such as workload-based partitioning [5]. The idea is to choose partitioning keys that “maximize”

the amount of EJ as opposed to RJ, by looking at the query workload. Additionally, for relations that are small in comparison to the data set that would need to be repartitioned to join with them, RRJ may be preferable [4], as it avoids potentially large repartitioning overheads. This is the reason why smallest relations (C and S) are P-replicated, as it avoids the need to ship larger data between nodes to join with those smaller data sets.

Query processing over a parallel database, and in particular over the NPDW, follows roughly the steps in Figure 3, which we describe in more detail in [6]. Figure 2 illustrates a simple example. Consider a sum query. Each node needs to apply exactly the same initial query (or more generically, a modified query) on its partial data, and the results are merged by applying a merge query again at the merging node with the partial results coming from the processing nodes.

More generically, the typical query processing cycle is shown in Figure 3 and a complete example is given in Figure 5. Step 1 prepares the node and merge query components from the original submitted query. Step 2 “Send Query” forwards the node query into all nodes in the NPDW, which process the query locally in step 3. Each node then sends its partial result into the submitter node, which applies the merge query in Step 5. Step 6 redistributes results into processing nodes if required (for some queries containing subqueries, in which case more than one processing cycle may be required).

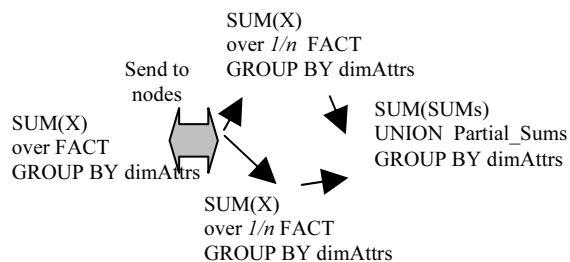


Figure 2– Typical Query over NPDW

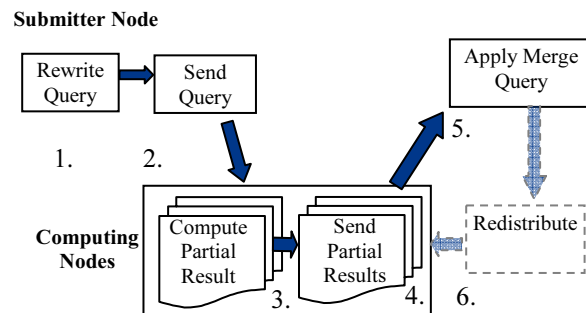


Figure 3 – Query Processing Steps in NPDW

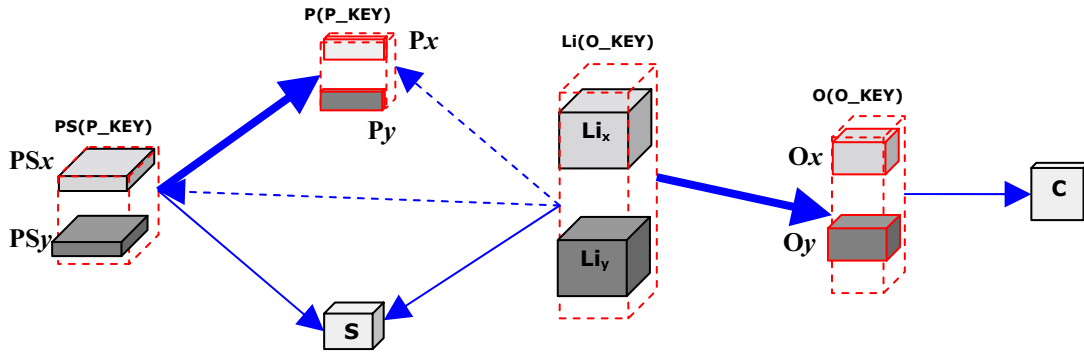


Figure 4: Schema in Node X with replicated Schema from Node Y

In steps 1 and 2 of Figure 4 we can see that Aggregation primitives are computed at each node. The most common primitives are:

- Linear sum: (LS=SUM(X));
- Sum of squares (SS=SUM(X²));
- number of elements (N);
- extremes (MAX and MIN).

0. Query submission:

```
Select sum(a), count(a), average(a), max(a), min(a),
stddev(a), group_attributes
From fact, dimensions (join)
Group by group_attributes;
```

1,2. Query rewriting and distribution to each node:

```
Select sum(a), count(a), sum(a x a), max(a), min(a),
group_attributes
From fact, dimensions (join)
Group by group_attributes;
```

3. Compute partial results:

```
Select sum(a), count(a), sum(a x a), max(a), min(a),
group_attributes
From fact, dimensions (join)
Group by group_attributes;
```

4. Results collecting:

```
Create cached table
PRqueryX(node, suma, counta, ssuma, maxa, mina,
group_attributes)
as <insert received results>;
```

5. Results merging:

```
Select sum(suma), sum(counta),
sum(suma) / sum(counta), max(maxa), min(mina)
(sum(ssuma)-sum(suma)2)/sum(counta), group_attributes
From UNION_ALL(PRqueryX), dimensions (join)
Group by group_attributes;
```

Figure 5 – Basic Aggregation Query Steps

Although we have discussed and evaluated extensively partitioning and processing choices for the NPDW in previous works, we did not discuss availability, which is nevertheless very important in the potentially unreliable environment for which NPDW is designed to run.

A discussion of availability for the NPDW brings up several issues. For instance: network failures; failure of the submitter or computing nodes; loading failures; availability monitoring, and so on. Each of these issues requires specific solutions. For instance, network failures can be accommodated using backup connections; unavailability of submitter node can be accommodated by allowing more than one node to be a potential submitter and routing client requests into available nodes; Failure of the submitter node in the middle of query processing can be handled by redirecting partial results into another node or resubmitting the query. These issues are part of our current and future work on the subject. In this paper we restrict our attention to the unavailability of computing nodes, replication alternatives to achieve high availability and processing efficiency in the presence of replication and unavailability.

4. Availability-targeted Replication over NPDW

Consider first that the basic replication unit in NPDW is the node. A whole copy of relation partitions from one node can be placed in another node and, in case of failure, the replacement node will process “twice” the amount of data – its own node data and the one it is replacing. In practice, P-replicated relations (small dimensions) do not need to be replicated again for availability. Figure 4 shows the schema of a node X with replicated data from another node Y. Node X can now replace node Y in case of unavailability of Y.

We will also discuss in the next section availability strategies that slice the replication units further and divide the slices by more than one node. This strategy improves the efficiency of processing in case of node unavailability. For instance, Li_y in Figure 4 will be replaced by Li_{yj}, j=1..m and divided into m nodes. In this case the unit of replication will be the slice.

There is also another requirement concerning replication slices. Consider a partition Li_i of a relation

Li that is placed at a node X. As depicted in Figures 1 and 4, relations are partitioned by a partitioning key (typically hash-partitioned) and placed in equi-partitioned fashion when possible (e.g. Li and O are both partitioned by O_KEY and tuples with a specific value of O_KEY are placed on the same node). The requirement is that replication slices also be organized by partitioning key in a similar way, so that tuples with the same key will still be co-located.

With respect to query processing with replicas, there are two issues: which nodes process which replicas and how they extend their processing to handle the replicas. The first issue is a scheduling problem which is not our main concern in this paper and for which we use a simple greedy solution:

```

Each node processes its own data;
For each unavailable node
    Choose replica-holding node with less load to
    process its replica
    (if more than one have same load, choose
    closest)

```

Although this algorithm does not guarantee balanced distribution of load, it is sufficient for our purposes and if there is imbalance in the result (e.g. the top load being a node with much more load than the other ones), a second step can try to reallocate the processing of one or more replicas from that node.

We now concentrate on how to handle replicas while processing queries. A node running a replica or slice can process its data set and the replica independently, as if it represented “two virtual nodes” running two independent instances of the cycle in Figure 3. These computations yield two partial results as if it were the partial results from two separate nodes, which can be merged using step 5 of Figure 3 before sending a single partial result to the merger node. The normal processing resumes as before in step 4, with every node sending their results to the merging node(s). This strategy is not the most efficient because the replacement node processes the whole data separately for both virtual nodes and applies an extra merge query. A better alternative is to scan the union of partitioned relations. Scan operations over partitioned relations now scan both the nodes’ data and the replicas’ data and the query proceeds as in a single node (with the query optimizer choosing the best query plan). This alternative is better because it avoids extra merging overhead and also the need to join twice with replicated relations that appears if the virtual nodes approach was used instead (one for each virtual node, while scan-union requires a single processing of replicated relations). As the scan union alternative is more

efficient than the virtual nodes approach, we adopted scan-union in NPDW (the experimental evaluation is based in scan-union).

5. Alternative Replication Strategies

In this section we consider and analyze alternative replication strategies. We analyze the advantages of each strategy using as metrics: degree of fault tolerance (how many nodes can be unavailable or fail simultaneously); efficiency (performance upon node failure); provision for taking several nodes offline simultaneously for data loading or other management or maintenance activities. For instance, it may be possible to take half the nodes offline for loading while the system remains online, then switch to loading the other half while never stopping the availability status of the system.

5.1. Full Replicas (FR)

The simplest replica placement strategy involves replicating each node’s data into at least one other node. In case of failure of one node, a node containing the replica resumes the operation of the failed node. A simple placement algorithm considering R replicas is:

```

Number nodes linearly;
For each node i
    For replica =1 to R
        data for node i is also placed in node (i+R) MOD N;

```

Metrics:

- Degree of fault tolerance: R nodes when considering R replicas;
- Efficiency (performance upon node failure): processing time doubles when a node fails;
- Provision for taking several nodes offline simultaneously: can take multiple nodes offline simultaneously, as long as the set of unavailable nodes does not include all R+1 copies of any node. For example, in Figure 6 with two replicas, shaded boxes may be unavailable and the system still works, because nodes 3, 6 and 9 contain replicas of their two closest neighbors. This suggests that up to $R/(R+1)N$ nodes can be offline simultaneously, if chosen carefully.

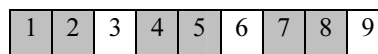


Figure 6: Availability in FR

The major drawback of this simple strategy is processing efficiency when unavailability of a few nodes occur: consider a NPDW system with N homogeneous nodes. Using a simplified linear model, assume that each node contains and processes about

1/N of the data in $O(1/N)$ of the time it would take to process the whole data. If one node fails, the node replacing it with the replica will take (at least) about twice as long $O(2/N)$, even though all the other nodes will take $O(1/N)$. The replica effort is placed on a single node, even though other nodes are less loaded.

5.2. Fully Partitioned Replicas (FPR)

Instead of having full replicas in a single node, much more efficiency results if replicas are partitioned into as many slices as there are nodes minus one. If there are N nodes, a replica is partitioned into N-1 slices and each slice is placed in one node. The replica of node i is now dispersed into all nodes except node i. The following algorithm can be used to place the slices:

Number nodes linearly;
 The data for node i is partitioned into N-1 numbered slices, starting at 1;
 For slice x from 1 to N-1:
 Place slice x in node $(i+x) \text{ MOD } N$.

This strategy is the most efficient one because, considering N nodes, each replica slice has $1/(N-1)$ of the data and each node has to process only that fraction in excess in case of a single node being unavailable. If a node becomes unavailable, the remaining nodes will process their data together with the replica slices corresponding to the unavailable node. However, in this case it is not possible to stop more than one node if there is a single replica, because all nodes that remain active are needed to process a slice from the replica. In order to allow up to R nodes to become unavailable, there must be R non-overlapping replica slice sets. Two replicas are non-overlapped iff the equivalent slices of the two replicas are not placed in the same node. Consider that R replicas are to be created (tolerance to unavailability of R nodes). In order to avoid slice overlapping, the following placement algorithm is used:

Number nodes linearly;
 The copy of the data of node i is partitioned into N-1 numbered slices, starting at 1.
 For j=0 to R:
 For slice x from 1 to N-1:
 Place slice x in node $(i+j+x) \text{ MOD } N$

Metrics:

- Degree of fault tolerance: R nodes, when R replicas are used;
- Efficiency (performance upon node failure): processing time increases proportionally to size of slice (fraction $1/(N-1)$);
- Provision for taking several nodes offline simultaneously: need multiple non-overlapping replicas.

5.3. Partitioned Replicas (PR)

Replicas may be partitioned into less than N slices (in NPDW with N nodes). If replicas are partitioned into x slices, we denote it by PR(x). If $x=N$, we have a fully partitioned replica. A very simple algorithm to generate less than N slices is:

Number nodes linearly;
 The data for node i is partitioned into X slices starting at 1;
 For slice set j=0 to R:
 For slice x from 1 to X:
 Place slice x in node $(i+j+x) \text{ MOD } N$

If we desire y nodes to be able to come offline simultaneously when a single replica is used, then the y nodes must not contain replica slices of each other. In order to achieve this, we can divide the nodes into groups that we want to take offline simultaneously. Then we guarantee by placement that replica slices of the nodes in a group are not placed in any node of that group and therefore we can take the whole group offline simultaneously for maintenance or other functionality.

For instance, Figure 7 shows twelve nodes organized into two groups G1 and G2. Replicas of each node are PR(6) and the slices are placed in the other group. The labels R1 and R2 in the Figure represent the replicas of nodes of each group and indicate that they are placed in the other group. The replicas are fully partitioned into the other group.

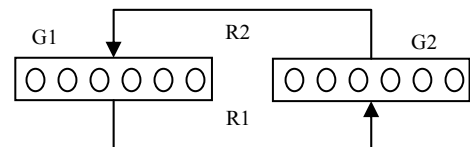


Figure 7: Grouping Replicas

Using this strategy, it is possible to take a whole group (6 nodes) offline simultaneously. The system will run slightly slower than if we had a single node offline with 12 full replica slices, because slices are larger. This layout guarantees availability to failures of a single node (R=1) but also of any number of nodes from a single group.

We denote this strategy by PRG(g,x) (g groups with x elements each) or PR(x), for simplicity and considering equal-sized groups. It works like FR at the inter-group level and FPR within each group. If we use this strategy with R replicas and R+1 groups, the system can tolerate failures or unavailability of nodes from up to R

groups. More groups allow more nodes to be unavailable but slices will be larger, leading to possibly slower processing when groups are offline.

Metrics:

- Degree of fault tolerance: X nodes from a single group; If R replicas over R+1 groups are used, the system can tolerate failures or unavailability of nodes from up to R groups;
- Efficiency (performance upon node failure): processing time increases proportionally to size of slice (fraction 1/(X));
- Provision for taking several nodes offline simultaneously: can take offline whole groups.

6. Comparative Analysis

In this analysis we focus on the balance between efficient availability, by analyzing the performance under node unavailability, and the flexibility to take multiple nodes offline. We consider the use of full replicas (FR), fully partitioned replicas (FPR) and partitioned replicas (PR). The analysis involved measuring response time of NPDW on low cost PCs (800MHz, 512 MB RAM). 50GB TPC-H [17] was manually setup into 1, 10 and 20 nodes, with partitioning and placement as described in section 3. We then measured response time for query 9 of TPC-H without nodes offline and compared the result to the response time with 5 nodes offline. Query 9 is reproduced below for reference (the query parameters were generated as described in the TPC-H specification and the results are the average of 10 runs).

```

Select nation, o_year, sum(amount) as sum profit from
(
Select n_name as nation, year(o_orderdate) as o_year,
l extendedprice * (1 - l discount) - ps_supplycost*
l_quantity as amount
from
tpcd.part,tpcd.supplier, tpcd.lineitem, tpcd.partsupp,
tpcd.orders, tpcd.nation
where
s supkey = l_supkey and ps supkey = l_supkey
and ps partkey = l_partkey and p_partkey = l_partkey
and o_orderkey = l_orderkey
and s_nationkey = n_nationkey
and p_name like x and n_nationkey > y
and o_orderpriority = 'z' and ps_availqty > w
) as profit
group by nation, o_year
order by nation, o_year desc;)

```

Figure 8 shows the response time (min:sec) when 5 out of 20 nodes are offline (line). The alternatives

compared are: “online” – every node is online; FPR – fully partitioned replicas, 5 nodes offline; PR(10) – partitioned replicas, two groups of 10 nodes each; PR(5) – partitioned replicas, 4 groups of 5 nodes each. It also shows the minimum number of replicas that are necessary to provide the required availability. These results show the much larger penalty incurred by FR and the excessive number of replicas required for FPR to allow 5 nodes offline simultaneously. PR(10) (partitioned replicas with two 10 element groups) are a good choice, as it requires a single replica and obtains a good response time simultaneously.

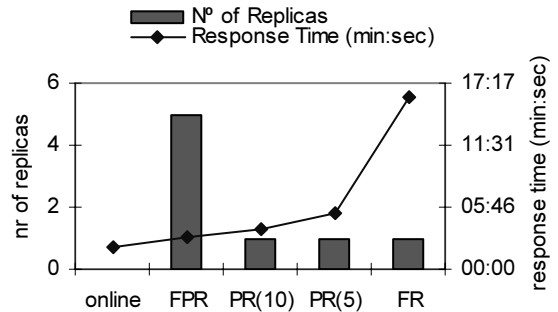


Figure 8: Response Time & Replicas 20 nodes/5 fail (Query 9)

The results for NPDW with 10 nodes are shown in Figure 9. In this case we consider 4 unavailable nodes instead of the 5 of the previous results and the pair PR(5), PR(2) instead of PR(10) and PR(5).

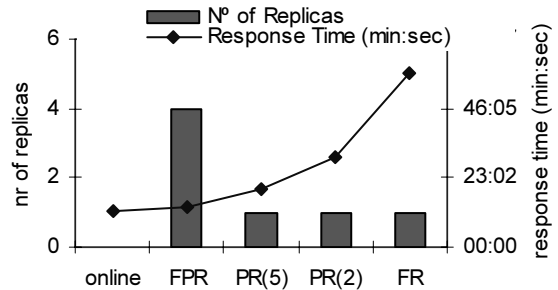


Figure 9: Response Time & Replicas 10 nodes/4 fail (Query 9)

The trend is similar to the one observed in Figure 8, the main difference being that the response times are much larger in every case because there are only half the number of nodes in Figure 9 (10 nodes in Figure 9 versus 20 nodes in Figure 8). In this case PR(5) seems to be the best choice, as it avoids the cost of FR or PR(2) and simultaneously the requirement of FPR that there be at least 4 replicas of each node.

Figure 10 compares the response time on NPDW with 10 nodes versus NPDW with 20 nodes. These results show that, although the response time with 10 nodes is much larger than that with 20 nodes, as

expected, the comparison between alternative replication schemes follows a similar trend.

These experimental results have shown that it is advantageous to consider partitioned replicas instead of simply full replicas if the system is to offer efficient availability. With such a capability, the system can be always-on, always efficient even though parts of it are taken offline for maintenance of management functions such as loading with new data or DBA functionality. We are currently testing the strategies over additional query workloads with varied characteristics.

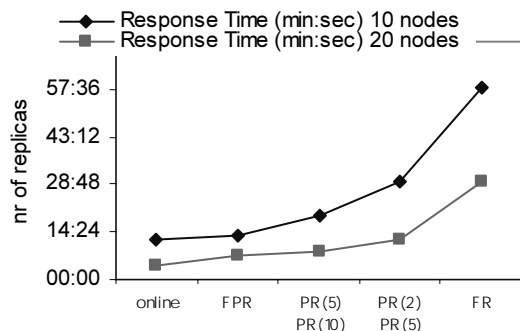


Figure 10: Comparison 10 nodes versus 20 nodes

7. Conclusions and Future Work

The work presented in this paper focused on replication for efficient availability on the Node Partitioned Data Warehouse (NPDW). After reviewing placement and processing issues over the NPDW, we have compared alternative replica strategies using metrics that included efficiency, degree of tolerance to node failures and capacity to allow multiple nodes to be offline simultaneously. The alternatives, ranging from full replication to various degrees of partitioned replication, were compared experimentally from the perspective of performance degradation when nodes go offline. We concluded that replicas partitioned by groups are the most advantageous alternative for NPDW if we consider both performance and flexibility in allowing multiple nodes to be taken offline simultaneously for maintenance or loading reasons. Besides extensive testing of the approaches, our future work in this subject includes automating replication and recovery, as well as automated data warehouse loading with the system always-on using the PR() strategies described in this paper.

8. References

[1] Coulon C., E. Pacitti, P. Valduriez, "Scaling up the Preventive Replication of Autonomous Databases in Cluster Systems", Vecpar 2004, 6th International Conference, Valencia, Spain, June 28-30, 2004.

[2] Copeland G., Tom Keller, "A comparison of high-availability media recovery techniques", In Procs of the 1989 ACM International Conf. on Management of Data.

[3] DeWitt D., Gerber R., "Multiprocessor Hash-Based Join Algorithms". Proceedings of the Eleventh Conference on Very Large Databases, 151-164, Stockholm, Sweden, August 1985.

[4] Furtado P., "The Issue of Large Relations in Node-Partitioned Data Warehouses", International Conference on Database Systems for Advanced Applications (DASFAA05), Beijing, China, April 2005.

[5] Furtado P., "Experimental Evidence on Partitioning in Parallel Data Warehouses", DOLAP 04 - WORKSHOP of the Int'l Conference on Information and Knowledge Management (CIKM), Washington, November-2004.

[6] Furtado P. "Efficiently Processing Query-Intensive Databases over a Non-dedicated Local Network". Nineteenth International Parallel and Distributed Processing Symposium, Denver, Colorado, USA, May 2005.

[7] Hsiao H., David J. DeWitt: Replicated Data Management in the Gamma Database Machine. Workshop on the Management of Replicated Data 1990.

[8] Hsiao H., David J. DeWitt: Chained Declustering: A New Availability Strategy for Multi-processor Database Machines. ICDE 1990.

[9] Hsiao H., David J. DeWitt: A Performance Study of Three High Availability Data Replication Strategies, PDIS 1991.

[10] Kimball, R. (1996). The Data Warehouse Toolkit. New York: J. Wiley & Sons.

[11] Kitsuregawa M., Tanaka H. and Motooka T., "Application of Hash to Database Machine and its Architecture", New Generation Computing, 63-74, 1(1).

[12] Lin Y., B. Kemme, R. Jimenez-Peris, "Consistent Data Replication: Is it feasible in WANs?" in 11th International Euro-Par Conference, Lisboa, Portugal, August 30-2, 2005.

[13] Pacitti E., M. Özsu, C. Coulon, "Preventive Multi-Master Replication in a Cluster of Autonomous Databases", 9th International Euro-Par Conference, Klagenfurt, Austria, August 26-29, 2003.

[14] Rao, J., Zhang c., Megiddo n., Lohman G, "Automating Physical Database Design in a Parallel Database", ACM International Conference on Management of Data, 558-569, Madison, Wisconsin, USA, June 2002.

[15] Tandem Database Group, "NonStop SQL, A Distributed, High-Performance, High-Reliability Implementation of SQL," Workshop on High Perform. Trans. Sys., CA, sept 1987.

[16] Teradata, "DBC/1012 Database Computer System Manual Release 2.0," C10-0001-02, Teradata, Nov 1985.

[17] TPC Benchmark H, Transaction Processing Council, June 1999. Available at <http://www.tpc.org/>.

[18] Yu, C. T. and Meng W. (1998). Principles of Database Query Processing for Advanced Applications. Morgan Kaufmann.

[19] Zilio, D. C., Jhingran A., Padmanabhan S., "Partitioning Key Selection for a Shared-Nothing Parallel Database System". IBM Research Report RC 19820 (87739), 1994.