

Redundant Array of Inexpensive Nodes for DWS

Jorge Vieira¹, Marco Vieira², Marco Costa¹, Henrique Madeira²

¹ Critical Software SA
Coimbra, Portugal
{jvieira, mcosta}@criticalsoftware.com

² CISUC, Department of Informatics Engineering, University of Coimbra
Coimbra, Portugal
{mvieira, henrique}@dei.uc.pt

Abstract. The DWS (Data Warehouse Striping) technique is a round-robin data partitioning approach especially designed for distributed data warehousing environments. In DWS the fact tables are distributed by an arbitrary number of low-cost computers and the queries are executed in parallel by all the computers, guarantying a nearly optimal speed up and scale up. However, the use of a large number of inexpensive nodes increases the risk of having node failures that impair the computation of queries. This paper proposes an approach that provides Data Warehouse Striping with the capability of answering to queries even in the presence of node failures. This approach is based on the selective replication of data over the cluster nodes, which guarantees full availability when one or more nodes fail. The proposal was evaluated using the newly TPC-DS benchmark and the results show that the approach is quite effective.

Keywords: Data warehousing, redundancy, replication, recovery, availability.

1 Introduction

A data warehouse (DW) is an integrated and centralized repository that offers high capabilities for data analysis and manipulation [8]. Data warehouses represent nowadays an essential source of strategic information for many enterprises. In fact, as competition among enterprises increases, the availability of tailored information that helps decision makers during decision support processes is of utmost importance.

Data warehouses are repositories that usually contain high volumes of data integrated from different operational sources. Thus, the data stored in a DW can range from some hundreds of Gigabytes to the dozens of Terabytes [7]. Obviously, this scenario raises two important challenges. The first is related to the storage of the data, which requires large and highly-available storage devices. The second concerns accessing and processing the data in due time, as the goal is to provide low response times for the decision support queries issued by the users.

In order to properly handle large volumes of data, allowing performing complex data manipulation operations, enterprises normally use high performance systems to

host their data warehouses. The most common choice is systems that offer massive parallel processing capabilities [1], [10], as Massive Parallel Processing (MPP) systems or Symmetric MultiProcessing (SMP) systems. Due to the high price of this type of systems, some less expensive alternatives have already been proposed [5], [6], [9]. One of these alternatives is the Data Warehouse Stripping (DWS) technique [2], [4].

In a simplified view, the DWS technique distributes the data of a data warehouse by a cluster of computers, providing near linear speedup and scale up when new nodes are added to the cluster [2]. However, adding nodes to the cluster also increases the probability of node failure, which in turn leads to a decrease in the cluster MTBF (Mean Time Between Failures). It is worth mentioning that a failure of a single DWS node is enough to render the whole DWS cluster out of service, as the execution of queries requires the availability of all data partitions [2].

Data redundancy is used in several applications to tolerate failures [12], [13]. In this paper we propose an approach that provides DWS clusters with high-availability even in the presence of node failures. The proposed approach, named RAIN (Redundant Array of Inexpensive Nodes), allows DWS to deliver exact query answers and is able to tolerate failures of several cluster nodes (the number of node failures tolerated depends on the configuration used). The RAIN technique is based on the selective replication of data over a cluster of low-cost nodes and comprises two types of replication: simple redundancy (RAIN-0) and striped redundancy (RAIN-S).

The proposal is illustrated using the newly TPC-DS benchmark [11]. Several configurations tolerating failures of different numbers of nodes have been tested. Results show that the performance in the presence of node failures is quite good.

The structure of the paper is as follows: section 2 briefly presents the DWS technique; section 3 describes the RAIN technique; section 4 presents the experimental evaluation; and Section 5 concludes the paper.

2 The DWS Technique

In the DWS technique the data of each star schema [2], [3] of a data warehouse is distributed over an arbitrary number of nodes having the same star schema (which is equal to the schema of the equivalent centralized version). The data of the dimension tables is replicated in each node of the cluster (i.e., each dimension has exactly the same rows in all the nodes) and the data of the fact tables is distributed over the fact tables of the several nodes using strict row-by-row round-robin partitioning or hash partitioning (see Fig. 1).

DWS data partitioning for star schemas balances the workload by all computers in the cluster, supporting parallel query processing as well as load balancing for disks and processors. The experimental results presented in [3] show that a DWS cluster can provide an almost linear speed up and scale up.

In a DWS cluster typical OLAP (OnLine Analytical Processing) queries are executed in parallel by all the nodes available. If a node of the cluster fails, the system is still capable of computing approximated answers to queries, which are computed by applying statistical formulas to the partial results obtained from the available nodes. The system is also capable of providing confidence intervals for the provided an-

swers. Obviously a degradation of the quality of answers is observed as the number of failed nodes increases.

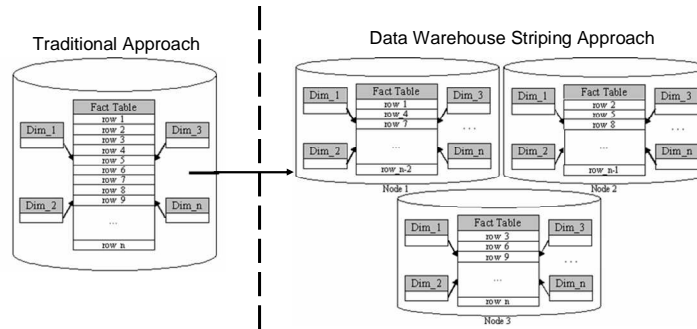


Fig. 1. Data Warehouse Striping Technique.

The approximate answers provided by DWS systems are normally useful for decision support, especially because the confidence intervals provide the user with a measure of the accuracy of the query results. However, in some business areas the approximated answers are not enough, as it is required that the system always provides exact answers to queries, even in the presence of node failures.

An important aspect is that the DWS technique allows enterprises to build large data warehouses at low cost. In fact, DWS can be built using inexpensive hardware and software and still achieve very high performance.

3 Redundant array of DWS nodes

In the DWS when a node fails the fact tables data stored in that node is no longer available, preventing the system to compute exact answers to queries. In order to overcome this problem, allowing the system to provide exact answers to queries in the presence of node failures, it is necessary to guarantee that the fact table data from a given node is still available in the cluster, even when that node fails.

The RAIN technique is based on the replication of fact table data across the cluster nodes. Two distinct approaches are proposed for the implementation of this technique: simple redundancy (RAIN-0) and stripped redundancy (RAIN-S).

Although the technique requires more disk space in each node, as the data of the fact tables of each node will be replicated in the other nodes (see details further on), the current disk sizes in inexpensive computers (the ones used in the DWS clusters) are large enough to accommodate the extra data space required. From our experience, an inexpensive machine (a typical PC) can process queries over a star schema up to 20 GBytes of fact data with acceptable response time. However, a typical PC has disks of 200 GBytes or more, which is more than enough to store the facts data of the node plus the portions of data replicated from other nodes (and the typical DW materialized views and indexes). That is, it really makes sense to trade disk space by system availability, as proposed in the RAIN technique.

3.1 Simple Redundant Array of Inexpensive Nodes

The simple redundancy approach consists of replicating the facts data from each node in other nodes of the cluster. Depending on the configuration used RAIN-0 can tolerate the failure of one or more nodes. The following notation is used to refer to a RAIN-0 cluster with N nodes that tolerates the failure of Y nodes: **RAIN-0(N, Y)**.

Fig. 2 presents the facts data replication schema in a cluster of five nodes designed to tolerate the failure of a single node: RAIN-0(5, 1). As shown, each node stores its own facts data (i.e., $1/5$ of the total facts data) plus the facts data from another node (i.e., each node stores a total of $2/5$ of the total facts data). This way if one of the nodes temporarily fails it is still possible to compute exact answers, since the data from that node also exists in another node.

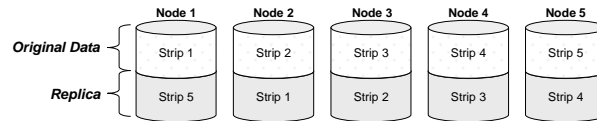


Fig. 2. Redundancy in Simple Redundant Array of Inexpensive Nodes designed to tolerate the failure of a single node: RAIN-0(5, 1).

It is important to emphasize that replicated data is used only in the case of node failures and no additional nodes are added to the cluster as that would represent a waste of computational resources in the absence of faults. Clearly, if more nodes are available they should be used during normal operation to increase the cluster performance and not only when there are failed nodes. To tolerate failures of several nodes the facts data must be replicated several times in different nodes. In fact, to tolerate the failure of N nodes the data must exist in $N+1$ nodes.

An important aspect is the overhead during the execution of queries in the presence of node failures. As the data from each node is replicated in at least another node, when a node fails one of the remaining nodes will have to process two times more data (i.e., a total of $2/N$ of all the data stored in the cluster, where N is the total number of nodes). For example, considering the RAIN-0(5, 1) cluster (see Fig. 2), when node 1 fails then node 2 must compute $2/5$ of the facts data while nodes 3, 4, and 5 have to compute $1/5$ of the facts data (as if there was no failure).

A basic rule is that to tolerate the failure of Y nodes a RAIN-0 cluster must comprise a minimum of $Y+1$ nodes. However, using the minimum number of nodes required implies that in the worst case scenario (failure of Y nodes) the single remaining node has to process all the facts data, which makes the system to act as a single server machine. Obviously, this is not the best approach and does not take advantage of the DWS data partitioning. This way, the minimum number of nodes recommended for a cluster should be equal to the number of simultaneous node failures to be tolerated multiplied by two: $N=Y*2$. This rule limits the maximum amount of data processed by any node to $2/N$ regardless of the number of failed nodes (that must obviously be inferior or equal to $N/2$).

3.2 Stripped Redundant Array of Inexpensive Nodes

The stripped replication is an evolution of the simple replication that intends to reduce the overhead caused on the queries execution when a node of the cluster fails. In this approach the facts data from each node is randomly distributed in $N-1$ sub-partitions (where N is the number of nodes) and each sub-partition is replicated in at least one of the other nodes (Fig. 3). As shown, each node stores $1/N + 1/N * 1/(N-1)$ of the facts data. The following notation is used to refer to a RAIN-S cluster with N nodes that tolerates the failure of Y nodes: **RAIN-S(N, Y)**.

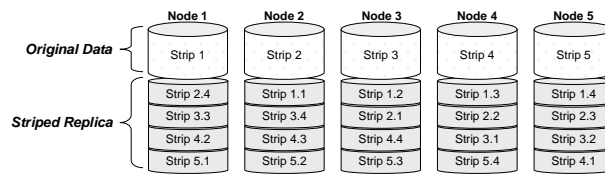


Fig. 3. Redundancy in Striped Redundant Array of Inexpensive Nodes designed to tolerate the failure of one node: RAIN-S(5, 1). Records from each partition are randomly distributed across four sub-partitions that are stored in the remaining four nodes.

Using RAIN-S replication strategy reduces drastically the overhead caused in the execution of a query when a node fails (when compared to RAIN-0). As the data from each node is evenly distributed by all the other nodes, when a node fails processing is distributed by all the remaining nodes. In other words, the remaining nodes have to compute only $(1/N) * 1/(N-1)$ more data (i.e., a total of $1/N + 1/N * 1/(N-1)$ of the all facts data stored in the cluster, where N is the total number of nodes). For example in a RAIN-S(20, 1) cluster each node will have to process approximately 0,2631% more data than it normally would: $(1/20) * 1/(20-1) = 0.2631\%$.

RAIN-S can also be configured to tolerate failures in several nodes. In this case, to tolerate the failure on Y nodes, each sub-partition must be replicated in at least Y nodes. An important aspect is that a RAIN-S cluster must have at least three nodes (with two nodes it is equal to a RAIN-0 cluster). Depending on the replication used, a cluster of 3 nodes is able to tolerate the failure of 1 or 2 nodes. Thus, the basic rule is that to tolerate the failure of Y nodes we need a RAIN-S cluster with $Y+1$ nodes. However, using the minimum number required nodes would mean that in the worst case scenario (failure of Y nodes) the single remaining node would have to process all the facts data, which is not the best approach as it does not take advantage of the DWS data partitioning. This way, the minimum number of nodes in a RAIN-S cluster should be equal to the number of simultaneous nodes failures to be tolerated multiplied by two: $N=Y*2$.

3.3 Execution middleware and node recovery

Query execution in DWS is enabled through the use of a middleware [2] that allow client applications (e.g., Oracle Discoverer, JPivot, Crystal Reports) to connect to the

system without knowing the cluster implementation details. This middleware receives queries from the clients, analyses the queries to be distributed by the nodes, submits the queries to the nodes, receives the partial results from the nodes and constructs the final result that is sent to the client application.

The DWS middleware was adapted to the RAIN technique, which means that it is prepared to transparently allow the system to continue providing exact queries answers when nodes fail. In fact, the middleware is able to detect failed nodes and uses the node’s replicas to compute the queries answers.

Node recovery is one problematic aspect when considering typical DWS systems. In fact, node recovery is not possible when stored data gets corrupt. The only way to recover a node is to recollect the data from the operational sources and rebuild the entire cluster. The use of RAIN eases the node recovery, as the recovery process can be accomplished using the data existing in the failed node replicas, without need of cluster downtime.

4 Experimental results and discussion

The goal of the experiments presented in this section was to measure the impact of node failures in the query execution when using RAIN-0 and RAIN-S approaches.

Table 1 presents the set of configurations tested. The basic platform used consists of seven Intel Pentium IV servers with 2GB of memory, a 120GB SATA hard disk, and running PostgreSQL 8.2 database engine over the Debian Linux Etch operating system. The servers were connected through a dedicated fast-Ethernet network.

Table 1. Set of cluster configurations used. All configurations include RAIN-0 and RAIN-S. The benchmark was executed two times for each configuration (a total of 24 runs).

RAIN	System	# Nodes	# Node Failures Tolerated
RAIN-0 & RAIN-S	(5, 0)	5	0
RAIN-0 & RAIN-S	(5, 1)	5	1
RAIN-0 & RAIN-S	(5, 2)	5	2
RAIN-0 & RAIN-S	(7, 0)	7	0
RAIN-0 & RAIN-S	(7, 1)	7	1
RAIN-0 & RAIN-S	(7, 2)	7	2

The PostgreSQL database is one of the most complete open-source databases available and is frequently used to support non-critical applications. For these reasons, we have chosen this DBMS as case-study for the validation of the RAIN technique.

The TPC Benchmark™ DS (TPC-DS) [11] is a performance benchmark for decision support systems. This benchmark evaluates the essential features of decision support systems, including queries execution and data load. The benchmark includes a set of seven stars with many dimensions and using snow-flakes and mini-dimensions. A scale factor must be defined, which allows the evaluation of systems of different sizes. In our experiments we used a database with 10GB (scale factor 10 in TPC-DS benchmark). Due to time constraints, a representative subset of the TPC-DS queries was used to perform the experiments. The queries were selected based on their intrinsic

sis characteristics and taking into account the changes needed for the queries to be supported by the PostgreSQL DBMS. Note that, as the goal is to evaluate the RAIN technique and not to compare the performance of the system with other systems, the subset of queries used is sufficient. The queries used are the following (see TPC-DS specification [11] for more details): 06, 07, 13, 15, 26, 27, 30, 37, 40, 45, 48, 52, 55, 75, 90, 91, 92, 96, 97, and 98.

Fig. 4 presents the execution times observed. As we can see, the RAIN-0(7, 0) presents an average execution time of 11.42 minutes, which is approximately 69% of the execution time observed for the RAIN-0(5, 0) configuration. This indicates a linear speedup during normal operation (without failures) of the cluster. Note that RAIN-0 and RAIN-S present the same execution time when the same number of nodes is considered (e.g., RAIN-0(5, 0) and RAIN-S(5, 0) have an execution time of 16,5 minutes). This is due to the fact that in these situations the cluster is operating in the absence of faults and thus the replication strategy does not influence the results.

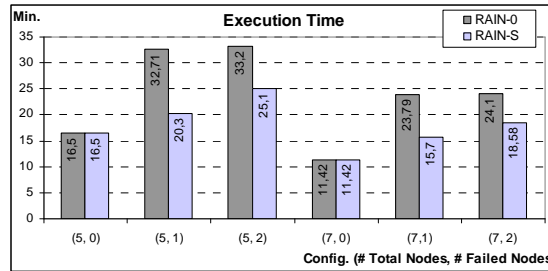


Fig. 4. Average execution time observed during the three executions of the benchmark in the several configurations tested.

As expected the cluster with seven nodes presents better results when compared to the cluster with five nodes. Comparing RAIN-0 and RAIN-S execution times, we can observe that the latter presents much better results. For example, for the cluster with 5 nodes the execution time when one of the nodes is down is 32.71 minutes for RAIN-0 and 20.3 minutes for RAIN-S. As explained before, this is due to the fact that for RAIN-0 one of the remaining nodes has to process all the data from the failed node while for RAIN-S the processing is distributed across the several nodes.

An important observation is that for RAIN-S the execution time increases slightly as the number of failed nodes increases.

As mentioned before, the main objective of DWS is to allow the construction of low cost data warehouse systems with acceptable performance. From our experience a DWS cluster with 10 nodes can process with acceptable performance queries for a data warehouse with a data volume of 200 GB. Considering off-the-shelf computers with the Intel Core 2 Quad Processor, 4GB of DDRII memory and two 160 GB SATAII hard drives, this cluster will have a retail price of approximately 11.000€.

The introduction of RAIN in a DWS cluster has a very low impact on the final price of the cluster. The cluster presented above has enough disk space to allow the use of RAIN technique without needing further hard drive devices, as each node has 320 GB of storage available and needs to store less than 40GB of data. Nevertheless we can add two extra hard drives on each cluster node for a cost of less than 1.000€.

This represents an increase in price of less than 10%, and allows using RAIN technique with a lower performance impact.

5 Conclusion

This paper proposes a technique that endows DWS clusters with the capability of providing exact queries answers even in the presence of node failures. The RAIN technique is based on the selective replication of data over the cluster nodes, which guarantees full availability when one or more nodes fail. Two types of replication are considered: simple redundancy (RAIN-0) and striped redundancy (RAIN-S).

The proposal was evaluated using the newly TPC-DS benchmark running on top of PostgreSQL and Debian Linux Etch operating system. The experimental results obtained were analyzed and discussed in detail. Results show that replication has no impact during the normal operation of the cluster. In addition, the performance impact in the presence of node failures is quite low.

Results show that by spending some disk space it is possible to achieve high availability with low performance degradation in the presence of node failures. This way, we believe that that the RAIN technique can be successfully applied to DWS clusters.

References

1. Agosta, L., "Data Warehousing Lessons Learned: SMP or MPP for Data Warehousing", DM Review Magazine, 2002.
2. Bernardino, J., Madeira, H., "A New Technique to Speedup Queries in Data Warehousing", ABDIS-DASFA, Symp. on Advances in DB and Information Systems, Prague, 2001.
3. Bernardino, J., Madeira, H., "Experimental Evaluation of a New Distributed Partitioning Technique for Data Warehouses", IDEAS'01, Grenoble, France, 2001.
4. Critical Software SA, "DWS", <http://www.criticalsoftware.com/>.
5. DATAlegro, "DATAlegro v3™", <http://www.datalegro.com/>.
6. ExtenDB, "ExtenDB Parallel Server for Data Warehousing", <http://www.extendb.com/>.
7. IDC, "Survey-Based Segmentation of the Market by Data Warehouse Size and Number of Data Sources", 2004.
8. Kimball, R., Ross, M., "The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (Second Edition)", Ed. J. Wiley & Sons, Inc, ISBN: 0471200247, 2002.
9. Netezza, "The Netezza Performance Server® Data Warehouse Appliance," <http://www.netezza.com/>.
10. Sun Microsystems, "Data Warehousing Performance with SMP and MPP Architectures", White Paper, 1998.
11. Transaction Processing Performance Council, "TPC Benchmark™ DS (Decision Support) Standard Specification, Draft Version 32", 2007, available at: <http://www.tpc.org/tpcds/>.
12. Lin, Y., et al., "Middleware based Data Replication providing Snapshot Isolation", ACM SIGMOD Int. Conf. on Management of Data. Baltimore, Maryland, USA, 2005.
13. Patino-Martinez, M., Jimenez-Peris, R., Alonso, G., "Scalable Replication in Database Clusters. International Conference on Distributed Computing", DISC'00, 2000.