An Adaptive Load Balancing Algorithm for Large Data Parallel Processing with Communication Delay

Kenji Imasaki and Jemal H. Abawajy

Carleton University, Ottawa, Canada,
{kenji,abawjem}@carleton.ca,
http://www.scs.carleton.ca/

Abstract. Achieving good load balance in cluster environment is difficult due to its dynamic nature. Besides, the data to be processed may be transfered from geographically distant sites at a fluctuating transfer rate. This paper proposes a load balancing algorithm to deal with such fluctuating transfer rate. The novelty of the algorithm is to use buddy concept in processing node grouping and the data distribution statistics obtained at the source sites. We use single join processing in database query processing as its application.

1 Introduction

Cluster computing technology is now fully blossomed due to the technology advances in processors, memory and high speed network. In a cluster, processing nodes, which are built from commodity products, are connected by LAN and heterogeneous in nature. There are interactive users who log-in/out from time to time so that it creates dynamic load changes and makes it difficult for applications to achieve its maximum performance.

Examples of such applications can be run on clusters are scientific computing and database processing. The input data for these applications such as measured data from sensors or XML structured database data, are large in size and may reside geographically remote site. Thus, the data has to be transfered to the cluster prior to the processing. The data transfer rate is unpredictable since it source site may be distant from the cluster and the connection between them may not be stable. At the same time, the applications should be parallelized because of their huge data sizes.

As a result, it is important to develop a good load balancing algorithm on a cluster to cope with data transfer rate fluctuation and dynamic load change within the cluster. The purpose of the algorithm is to minimize the response time and maximize the system utilization. In this paper, we will present an load balancing algorithm to deal with fluctuating data transfer rate and dynamic load changes. The novelty of the algorithm is to use buddy concept in processing node grouping and the data distribution statistics obtained at the source sites. The

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2660, pp. 242–247, 2003.

data is very helpful in the load balancing decision. We use single join processing in database query processing as its application.

This paper is organized as follows: Section 2 describes related work. Section 3 presents our load balancing algorithm with system architecture description. Section 4 will conclude the paper.

2 Related Work

The fluctuating data transfer rate problem is tackled in database query processing area. Several data integration systems [2] [3] [4] [5], which process queries over geographically distributed databases, have been developed. Most of the research have been concentrating in developing algorithms to deal with the fluctuating data transfer rate. Amsaleg et al. [1] proposed query scrambling in which the sub-query are "scrambled" according to the arrival rate of the data. Urhan [12] proposed XJoin in which several kinds of joins are executed while arriving data is stalled.

Single join processing, which is dealt in this paper, is a classic problem in the database query processing. Hash join based algorithms are the best algorithm [11]. To deal with data skew, several load sharing/balancing algorithms are proposed. Their comparison is summarized by Hua et al.[6].

Imasaki et al. proposed adaptive single hash join algorithms [7] [8] and multiple join algorithms [9] [10] on a workstation cluster. Their algorithms are called chunk-HJ which combines a chunking method with hash join to manage dynamic changes that occur in cluster environment.

3 Load Balancing Algorithms

This section first explains the query processing architecture that we used and then introduces the load balancing algorithm on the architecture.

3.1 The Query Processing Architecture

The query processing architecture which is dealt in this paper is shown in Figure 1. We assume processing relations are distributed over the Internet. A Database Management System (DBMS) is running on a remote site and reads relations from the database according to users' requests.

We use a cluster (called main cluster) to process queries submitted by users within the cluster. The main cluster with a large number of processing nodes is responsible for the query processing. The Query Manager (QM) runs on one of the processing nodes and is responsible for executing queries. The Join Manager (JM) is responsible for executing joins. Processing nodes (P) within the main cluster are divided into several *buddies*. The JM decides the work distribution among the buddies. The function of the buddy will be explained later in detail. The buddy manager (BM) is invoked for each buddy and is responsible for work distribution among processing nodes within the buddy.

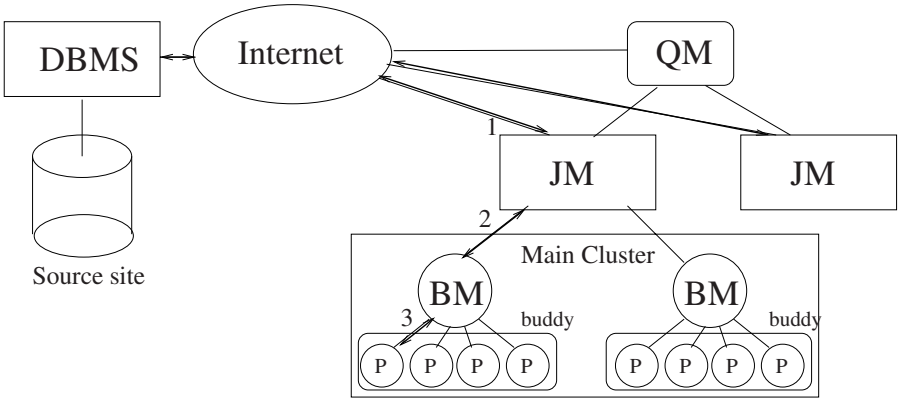

Fig. 1. The Query Processing Architecture

3.2 The Proposed Algorithm

The proposed algorithm for single join processing is explained by using the notation given in Table 1. The algorithm is divided into three parts as follows:

Interaction between source site and QM/JM ('1' in Figure 1)

1. Source sites ($DBMS_R$ and $DBMS_S$) start reading R^i and S^i, apply predetermined hash function to them and get R_h and S_h.
2. After reading *sampleSize* of R and S, DBMSs start sending the bucket distribution statistics to JM.
3. JM decides the BG assignments from R_h and S_h using bucket distribution statistics and load balancing consideration and let the source sites know the decision.
4. Source sites ($DBMS_R$ and $DBMS_S$) send buckets which belongs to the same BG together to JM.

Interaction between JM and BM ('2' in Figure 1) JM controls the load balancing among BMs by changing the number of processing nodes for each buddy at a certain interval based on the amount of work the buddy will process in future. There are two kinds of such statistics: $BG_x^{arrived}$ and BG_x^{source}. Their relationship is shown in Figure 2, together with BG_x^{past} in the case of the number of BGs is 4. In this figure, the number of tuples that belongs to BG_1 ($BG_1^{arrived}$) is small right now. However, it will increase in the future as BG_1^{source} is rather large. The amount of this time delay for the increase depends on the transfer speed of the network between the source site and the main cluster.

Thus, the number of processing nodes for BG_x (p_x) is decided assuming p processing nodes are available as follows:

$$p^{arrived} = x * p$$

Table 1. Notation used in algorithm description

R,S	two relations involved in the join
$DBMS_R$	DBMS for relation R
$DBMS_S$	DBMS for relation S
R^i	ith R segment
R_h	hth R hash bucket after applying the hash function
BG_x	a bucket group which consist of several R/S buckets
BG_x^i	ith segment of BG_x; the unit for work allocation
sampleSize	sampling size
p	the total number of available processing nodes
p_x	the number of processing nodes for a buddy
$p^{arrived}$	the total number of processing nodes for arrived data
p^{source}	the total number of processing nodes for source data
$p_x^{arrived}$	the number of processing nodes for a buddy for arrived data
p_x^{source}	the number of processing nodes for a buddy for source data
BG_x^{past}	the processed part of BG_x
$BG_x^{arrived}$	the arrived part of BG_x
BG_x^{source}	BG_x at the source site

$$p^{source} = (1 - x) * p$$

$$p_x = (BG_x^{arrived})/(\sum_i BG_i^{arrived}) * p_x^{arrived} + (BG_x^{source})/(\sum_i BG_i^{source}) * p_x^{source}$$

where x is the constant and and it is set between 0 to 1; x is set to 0 when source site is very close to the main cluster and/or the network speed is very fast; x is set to 1 in the opposite case.

With the new p_x, JM adjusts the number of processing nodes for each buddy.

Load Balancing Within Buddy ('3' in Figure 1) The algorithm within the buddy can be any parallel single join algorithm. We use the chunking-HJ proposed by Imasaki et al.[8]. It has robust performance that takes into account the following factors: data size, degree of data skew, degree of background load, or any combination of these. The algorithm does not require any knowledge of distribution and it does not involve pre-processing of the input relations.

4 Concluding Remarks

This paper proposed an adaptive load balancing algorithm for query processing with large communication delay. The algorithm uses buddy concept and data distribution statistics obtained at source sites. As we described in Introduction, this algorithm can be used in not only database processing but also other applications dealing with large amount of data.

We are currently evaluating the algorithm using an event-driven simulation programs. The fluctuating transfer rate is simulated by collected trace data.

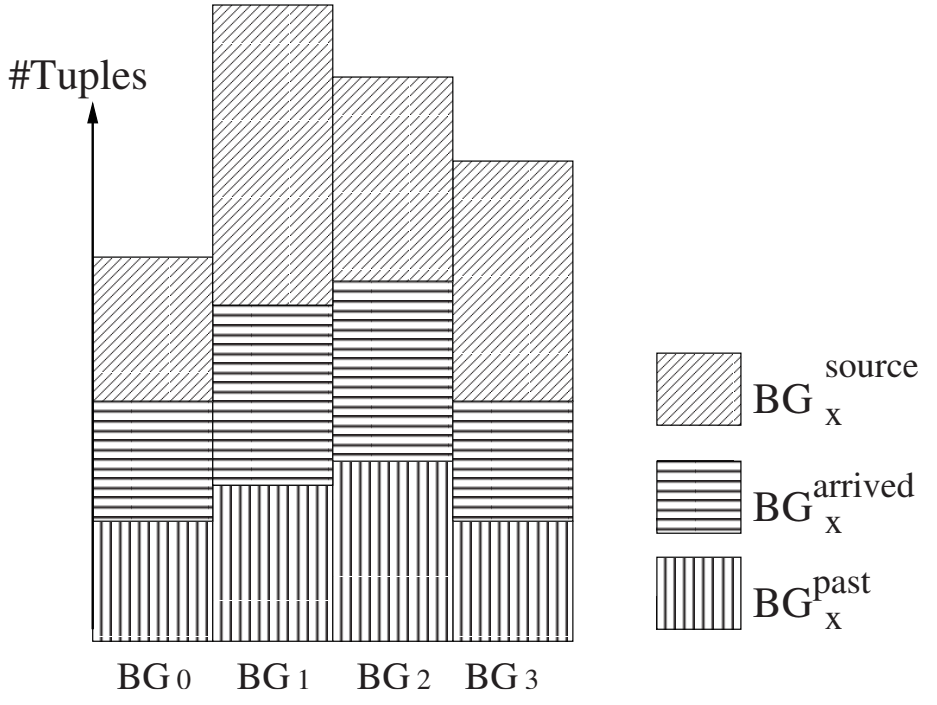

Fig. 2. Bucket Group Distribution in the case its number is 4

References

1. L. Amsaleg, M. J. Franklin, A. Tomasic, and T. Urhan. Scrambling Query Plans to Cope with Unexpected Delays. In *Fourth International Conference on Parallel and Distributed Information Systems (PDIS '96)*, pages 208–219, Miami Beach, Florida, USA, Dec. 1996. IEEE Computer Society Press.
2. R. Avnur and J. M. Hellerstein. Eddies: Continuously Adaptive Query Processing. In W. Chen, J. Naughton, and P. A. Bernstein, editors, *The 2000 ACM SIGMOD International Conference on Management of Data*, volume 29(2) of *SIGMOD Record (ACM Special Interest Group on Management of Data)*, pages 261–272, Dallas, Texas, May 2000. ACM Press.
3. L. Bouganim, F. Fabert, C. Mohan, and P. Valduriez. A Dynamic Query Processing Architecture for Data Integration Systems. In *16th International Conference on Data Engineering*, pages 425–434. IEEE Computer Society Press, Mar. 2000.
4. L. Bouganim, F. Fabert, C. Mohan, and P. Valduriez. Dynamic Query Scheduling in Data Integration Systems. *IEEE Data Engineering Bulletin: Special Issue on Adaptive Query Processing*, 23(2):42–48, June 2000.
5. J. M. Hellerstein, M. J. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. Shah. Adaptive Query Processing: Technology in Evolution. *IEEE Data Engineering Bulletin: Special Issue on Adaptive Query Processing*, 23(2):7–18, June 2000.

6. K. A. Hua and W. Tavanapong. Performance of Load Balancing Techniques for Join Operations in Shared-nothing Database Management Systems. *Journal of Parallel and Distributed Computing*, 56(1):17–46, Jan. 1999.
7. K. Imasaki and S. Dandamudi. Performance Evaluation of Nested-loop Join Processing on Networks of Workstations. In *Proceedings of the Seventh International Conference on Parallel and Distrubuted Systems*, pages 537–544, Iwate, Japan, July 2000.
8. K. Imasaki and S. Dandamudi. An Adapive Hash Join Algorithm on a Network of Workstations. In *International Parallel and Distributed Processing Symposium (IPDPS)*, Fort Lauderdale, Florida, Apr. 2002.
9. K. Imasaki, H. Nguyen, and S. Dandamudi. Performance Comparison of Pipelined Hash Joins on Workstation Clusters. In *9th International Conference on High Performance Computing (HiPC2002)*, pages 264–275, Bangalore, India, Dec. 2002.
10. K. Imasaki, H. Nguyen, and S. Dandamudi. Performance of Pipelined Nested Loop and Hash Joins on a Workstation Cluster. In *The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, June 2002.
11. D. A. Schneider and D. J. DeWitt. A Performance Evaluation of Four Parallel Join Algorithms in a Shared-nothing Multiprocessor environment. *SIGMOD Record*, 18(2):110–121, June 1989.
12. T. Urhan and M. J. Franklin. XJoin: Getting Fast Answers From Slow and Bursty Networks. Technical Report CS-TR-3994, University of Maryland, College Park, Feb. 1999.