High Performance Dynamic Load Balancing with Inter-Dependent Tasks in Heterogeneous Databases

Pritesh G. Shah
Department of Computer Science
Mahatma Gandhi Shikshan Mandal's
Arts, Science and Commerce College, Chopda, Dist: Jalgaon (M.S)

ABSTRACT
While data mining has its roots in the traditional fields of machine learning and statistics, the total volume of data mostly poses the most serious problem for which many organizations have data warehouses. Implementation of data mining ideas in high-performance parallel and distributed computing environments is thus becoming crucial for ensuring system scalability and interactivity as data continues to grow relentlessly in size and complexity. The large set of evolving and distributed data can be handled efficiently by Parallel Data mining and Distributed Data Mining. In this paper we present a load balancing techniques that can deal with inter dependent task. Instead of balancing the load in cluster by process migration, or by moving an entire process to a less loaded computer, we make an attempt to balance load by splitting processes into separate jobs and then balance them to nodes.

Keywords
Heterogeneous cluster, Dynamic load balancing, distributed systems, Parallel data mining, Distributed data mining.

1. INTRODUCTION
Data Mining and Knowledge Discovery in Databases (KDD) is a new interdisciplinary field merging ideas from statistics, machine learning, databases, and parallel and distributed computing. It has been engendered by the phenomenal growth of data in all spheres of human endeavor, and the economic and scientific need to extract useful information from the collected data. The key challenge in data mining is the extraction of knowledge and insight from massive databases.

Data mining refers to the overall process of discovering new patterns or building models from a given dataset. There are many steps involved in the KDD enterprise which include data selection, data cleaning and preprocessing, data transformation and reduction, data-mining task and algorithm selection, and finally post-processing and interpretation of discovered knowledge [1,2]. This KDD process tends to be highly iterative and interactive.

Typically data mining has the two high level goals of prediction and description [1]. In prediction, we are interested in building a model that will predict unknown or future values of attributes of interest, based on known values of some attributes in the database. In KDD applications, the description of the data in human-understandable terms is equally if not more important than prediction. Two main forms of data mining can be identified [3]. In verification-driven data mining the user postulates a hypothesis, and the system tries to validate it.

The common verification-driven operations include query and reporting, multidimensional analysis or On-Line Analytical Processing (OLAP), and statistical analysis. Discovery-driven mining, on the other hand, automatically extracts new information from data, and forms the main focus of this survey. The typical discovery-driven tasks include association rules, sequential patterns, classification and regression, clustering, similarity search, deviation detection, etc.

The rest of this paper is organized as follows. First section gives the introduction and some related terms, followed by second section which provides the system overview. Third section describes the proposed load balancing algorithm followed by load balancing strategies. Fourth section describes the mechanism for load balancing. Finally, we conclude this paper.

1.1 Parallel and Distributed Data Mining
Parallel data mining (PDM) deals with tightly-coupled systems including shared-memory systems (SMP), distributed-memory machines (DMM). Distributed data mining (DDM), on the other hand, deals with loosely-coupled systems such as a cluster over a slow Ethernet local-area network.

PDM is the ideal choice in organizations with centralized data-stores, while DDM is essential in cases where there are multiple distributed datasets.

Parallel and distributed computing is expected to relieve current mining methods from the sequential bottleneck, providing the ability to scale to massive datasets, and improving the response time.

One of the main challenge on which we are focusing in this paper include work-load balancing, which is especially important for data mining where other challenges include finding good data layout and data decomposition, and disk I/O minimization.

1.2 Static vs. Dynamic Load Balancing

There are many approaches to balancing load in disk I/O resource can be found in literature [4][5][6][7][8][9].

In static load balancing work is initially partitioned among the processors using some heuristic cost function, and there is no subsequent data or computation movement to correct load imbalances which result from the dynamic nature of mining algorithms. Dynamic load balancing seeks to address this by stealing work from heavily loaded processors and re-assigning it to lightly loaded ones.

Computation movement also entails data movement, since the processor responsible for a computational task needs the data associated with that task as well. Dynamic load balancing thus incurs additional costs for work/data movement, but it is beneficial if the load imbalance is large and if load changes with time. Dynamic load balancing is especially important in multi-user environments with transient loads and in heterogeneous platforms, which have different processor and network speeds. These kinds of environments include parallel servers, and heterogeneous, meta-clusters.

Dynamic load balancing algorithms make changes to the distribution of work among workstations at run-time; they use current or recent load information when making distribution decisions.

2. SYSTEM OVERVIEW

In this study we have considered a cluster computing platform of heterogeneous system in which n nodes are connected through high speed network. Where each node is a combination of various resources including processor, memory, disk, network connectivity.

A load handler is responsible for load balancing and monitoring available resources of the node. Fig 1 shows the queuing system for load handler.

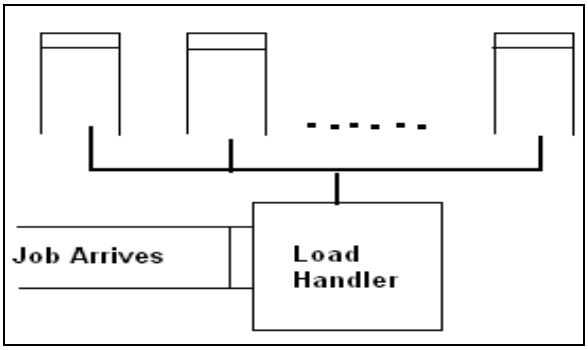

Fig 1: Queuing System for load handler

Load manager or master node process all arrival task in a FCFS manner. Here we are assuming that all tasks are to load manager is poison process. After being handled by load manager task are dispatched to one of the best suited node for execution.

The nodes, each of which contains a local queue, can execute task in parallel. Load manger is composed of three modules: (1) predictor; (2) selector; (3) scheduler; When new task is arrives at load manager; the identification of program being executed is sent to the predictor which predicts the resource requirements of the task.

These predicted values are then fed to the selector which selects the node with underutilized and well suited for its requirement resource. Predictor is used to predict the file I/O, CPU and memory requirements of a task. For this we use prediction scheme described in [7] uses a statistical pattern-recognition to predict the task requirements. The prediction is made at the beginning of a process's life, given the identity of the program being executed.

3. LOAD BALANCING ALGORITHM

We proposed an algorithm for a wide variety of workload conditions including I/O-intensive, CPU-intensive and memory intensive load. The objective of the proposed algorithm is to balance the load of three types of resources across all nodes in a cluster. In this study analytically evaluate the performance of algorithm; we are focused on a remote execution mechanism in which task can be running on a remote node where it started execution. Thus preemptive migrations of tasks are not supported in our algorithm.

To describe this algorithm first we introduce the following three load indices with respect to Input/output, CPU, memory resources. (1) CPU load of a node is characterized by the length of CPU waiting queue. (2) Memory load of a node is the sum of the memory space allocated to the entire task running on that node. (3)Input/output load measures two types of Input/output accesses.

Now we describe the load balancing algorithm of which the pseudo code is shown in Fig.2. Given a set of independent tasks submitted to the load Handler.

Input a job with task j given to the load handler

1. for each task
2. Calculate: Memory requirements, Input/output, CPU.
3. IRj= max(IRj, CRj, MRj)
4. Calculate response time R of task j in set of nodes.
5. Calculate whose response time is minimum, execute at that node after dispatching the task.
6. Change the current load.
7. end for
8. Loop
9. Wait for load change
10. if(activity_done())
11. if(load_balancing_start())
12. while Heavily_loadedlist is not empty
13. Determine the tasks to be migratable such that having minimum CPU consumption.

14. selected job=j

15. If Lightly_loadedlist is empty

16. Remaining_jobs=remaining_jobs+1

17.Else

18.Migrate(Lightly_loadedlist[first], Heavily_loadedlist[n], j, R)

19. End While

20. End Loop.

activity_done() returns boolean value. It returns true if any one of this activity occurs otherwise it returns false.

load_balancing_start() also returns Boolean value. It returns true if load balancing is required based on some parameters. Otherwise it returns false.

Whenever any of activities occur, it starts collecting load balancing information after which it is decided that whether load balancing information is required or not.

Our algorithm make an effort to balance the load of the cluster resource's by allocating each task to a node such that the expected response time is minimized

3.1 Load Balancing Strategies

There are three major parameters which usually define the strategy a specific load balancing algorithm will employ, which are important in order to address issues such as who makes the load balancing decision, what information is used to make the load balancing decision and where the load balancing decision is made.

We concentrate on selecting a policy such that in policy selection, information gathering policy specifies the strategy for the collection of load information including the frequency and method of information gathering. Information policy specifies what workload information to be collected, from where it is to be collected. The frequency is determined based on a tradeoff between the accuracy of load information and the overhead of information collection.

Initiation Policy determines who starts the load balancing process. The process can be initiated by an overloaded server (sender-initiated) or by an under-loaded server (receiver-initiated). Sender initiated policies are those where heavily loaded nodes search for lightly loaded nodes while receiver initiated policies are those where lightly loaded nodes search for suitable senders.

Job Transfer Policy determines when job reallocation should be performed and which job(s) should be reallocated. Job reallocation is activated by a threshold based strategy. In a sender-initiated method, the job transfer is invoked when the workload on a node exceeds a threshold. In a receiver-initiated method, a node starts the process to fetch jobs from other nodes

when its workload is below a threshold. The threshold can be a pre-defined static value or a dynamic value that is assessed at runtime based on the load distribution among the nodes. When job reallocation is required, the appropriate job(s) will be selected from the job queue and transferred to another node.

Resource type policy classifies a resource as a server or receiver of a task according to its availability status. Location policy uses the results of resource type policy to find who work co-ordingly with server or receiver. Selection policy defines the tasks that should be migrated from overloaded resources to most idle resources.

4. MECHANISM FOR LOAD BALANCING

For load transfer among different nodes each node maintains its own list of participating nodes to which it wants to communicate for load sharing. Each node maintains its own job queue. We have to consider interval between arrival time of task and the time at which last task was executed. For that At the beginning of each time interval, each node calculates its load from previous interval which can be designated as difference between the load.

Since different node may different intervals at any given time. It calculates the number of time intervals it will take to reach an idle state (no tasks to process). If the number of

intervals are less than the network delay then the node will initiate a migration request. The node responsible for initiation of load transfers and performs the actions such that initially select the node from which message will be sent, assign it to mode 'W' which stands for Waiting state so that it will not further issue any request until replay comes.

Considering this load request and transfer between the nodes[10]. Each node keeps two local tables containing system load information. One contains information regarding the location of sink Nodes (under loaded nodes), called sink table, the other

of source Nodes (overloaded nodes), called source table. Any node that initiates a request for load is considered to be a sink by the receiving node(s). The sink node (request initiator) selects a source node from its source table (the first entry in the table) and sends a message, requesting for load transfer to it. Initially the table is empty since no information is available regarding the state of the node is known and therefore a node is chosen at random. It only means that when no information is known regarding the load of any node in the system then every node is as likely to be considered a source node as any other and therefore we chose one among all possible ones at random.

The principle of load balancing algorithm is to speed up the execution of applications on resources whose workload varies at runtime such that we can't identify it. Every dynamic load balancing approach must estimate the timely workload information of each resource.

*Modification to the above algorithm involves calculating the load on clusters and then implementing the load balancing with inter-dependent tasks so that minimum response time considering inter-agent communications

such that the framework for load balancing consisting of multi-agent with each agent has a specific role to play and have facility for inter agent communication where each agent is implemented for managing hosts processors of a Cluster resource and scheduling incoming tasks to achieve load balancing. The various layers are: Communication and Coordination Layers, Management Layer.

5. CONCLUSION

The proposed load balancing scheme aim to achieve the effective usage of global disk resources in cluster. This can minimizes the average slow down of all parallel jobs running on a cluster and reduce the average response time of the jobs.

Even though there are number of different dynamic load balancing techniques for cluster systems, their efficiency depends topology of the communication network that connects nodes. This research has developed an efficient load balancing so that there is effective simulation between inter-dependent tasks.

Two limitations of the load balancing algorithm include: Preemptive migrations of tasks are not supported, and simulate this scheme for inter-dependent task. These two drawbacks are removed in the proposed modified* load balancing algorithm.

6. ACKNOWLEDGMENTS

Author wishes to thanks my Parents, Dr. Suresh G. Patil, Founder President, Adv. Sandeep Suresh Patil, President, Dr. Smita S. Pati,l Secretary, MGSM, Chopda, Dr. D. D Patil, Principal, Dr. A.L. Chaudhari Head of Computer & Electronics department, Mr. V.T. Patil Head of Physics department, Arts, Science & Commerce College, Chopda Dist. Jalgaon, Maharashtra India for giving the cooperation during the research work.

7. REFERENCES

[1] Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From data mining to knowledge discovery: An overview. [86]

[2] Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: The KDD process for extracting useful knowledge from volumes of data. Communications of the ACM **39** (1996)

[3] Simoudis, E.: Reality check for data mining. IEEE Expert: Intelligent Systems and Their Applications **11** (1996) 26-33

[4] Xiao Qin, Performance comparisons of load balancing algorithms for IO-intensive workloads on clusters, Journal of Network and computer applications(2006), doi:10.1016/j.jnca.2006.07.001

[4] Xiao Qin ,Dynamic Load Balancing for IO-Intensive Tasks on Heterogeneous Clusters, Proceeding of the 2003 International Conference on High Performance Computing(HiPCO3)

[5] Xiao Qin ,Hong Jiang ,Yifeng Zhu ,David R. Swanson ,A Dynamic Load Balancing Scheme for IO-Intensive Applications in Distributed Systems, Proceeding of 2003 international conference on Parallel processing Workshop(ICPP 2003 Workshop)

[6] Xiao Qin, A feedback control mechanism for balancing I/O intensive and memory-intensive applications on cluster, parallel and distributed computing practices journal

[7] Xiao Qin, H.Jiang, Y.Zhu and D.swanson, Toward load balancing support for I/O intensive parallel jobs in a cluster of workstation, Poc. Of the 5th IEEE international conference cluster computing(cluster 2003) ,Hong Kong, Dec. 1-4-2003

[8] M. Kandaswamy, M.Kandemir, A.Choudhary, D.Benholdt, Performance implication of architectural and software techniques on I/O intensive application, Proc International conference parallel processing 1998

[9] Kumar K. Goswami, Murthy Devarakonda and Ravishankar K. Iyer, Prediction–baesd dynamic load-sharing heuristics, IEEE transaction on parallel and distributed systems, VOL.4, No.6, june 1993

[10] Xiao Qin, Performance comparisons of load balancing algorithms for IOintensive workloads on clusters, Journal of Network and computer applications(2006), doi:10.1016/j.jnca.2006.07.001.