Optimization for Distributed Committee Machines in The Knowledge Discovery in Distributed Databases Process

Valentin Pupezescu

Electronics, Telecommunications and Information Technology Faculty, Polytechnic University of Bucharest
Blvd. Iuliu Maniu, Bucharest, ROMANIA
E-mail: vpupezescu@yahoo.com

Abstract
The knowledge discovery in distributed databases represents the overall process of obtaining useful information from data that is stored in a replication topology on multiple computing systems. Distributed committee machines are formed by many neural networks that work in a distributed manner on multiple computing systems for resolving one data mining task. These architectures are the most suitable neural structures for working in a replication topology. In my case I used multilayer perceptrons trained with the backpropagation algorithm for resolving the classification task. In this paper I propose an optimized way of functioning for the entire distributed committee machine architecture by choosing the appropriate database engine, the best type of replication and by reducing the writing operations into the distributed database as much as possible. All the experiments were done on large data sets in order to achieve the best results. This study is useful in research areas such as distributed learning, adaptive eLearning, machine learning, medical diagnosis, astronomy and many others.

Keywords: Knowledge Discovery in Distributed Databases, Data Mining, Distributed Learning

1 Introduction
"Knowledge Discovery in Databases (KDD) is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data" (Fayadd, U., et al, 1996). It consists of the following steps: selection of target data, preprocessing of the raw data, transformation of data, data mining and the interpretation(evaluation) of the obtained results (Fayadd, U., et al, 1996). If the obtained results are not useful, the whole process must be repeated.

One of the most important phase in the entire KDD process is the Data Mining(DM) step. Usually DM tasks are: classification, regression, clustering, summarization, dependency modeling, change and deviation detection.

Knowledge Discovery in Distributed Databases(KDDD) is finding useful information in distributed databases. If data is distributed across multiple computing machines, is important to have a DM architecture that suits well for working in such a manner. Based on my research I chose to work with Distributed Committee Machines(DCM) for resolving the classification task.

2. Distributed Committee Machines
The Distributed Committee Machine architectures consists of many neural networks(I used multilayer perceptrons) that work in a distributed manner in order to achieve better results than only one neural structure.

In previous work I proposed an optimized design of a real implementation of DCM architecture that is working with distributed databases(Pupezescu V., 2015):
I discovered that a DCM implementation that uses a real distributed database (MySql) works very slow if neural networks are writing on the master server in the table results. If the table results is replicated on all distributed systems we will obtain distributed accelerations that are almost zero (Pupezescu V., 2015). In this design I proposed that just input data to be replicated on all distributed systems (in this way we will have the same input data for all the neural networks). Each distributed result is written locally. In the end, the combiner will pick or combine the individual results from each distributed slave. Another optimization was the storing of the neural networks as BLOB (Binary Large Objects). In this way I have the advantage of keeping the whole neural structures as Java object in database so it will not be necessary to memorize all the weights in separate files.

I created the result table on the combiner with the following structure:

\[
\text{Result}(\text{id, \ PCICtest, \ Pi, \ Po, \ ip, \ problem}).
\]

I used the following data types for the attributes: \(\text{id} - \text{BIGINT}(20)\) (this field is the PRIMARY KEY for the table), \(\text{PCICtest} - \text{DOUBLE}\) (or \(\text{PCICg}\) - this field represents the percentage of incorrect classifications in tests - how many input data from the test set are incorrect classified), \(\text{Pi} - \text{BLOB}\) (this is the entire multilayer perceptron stored as Java object - this is the initial state of the multilayer perceptron), \(\text{Po} - \text{BLOB}\) (this is the optimum state of the same multilayer perceptron that was trained - in this form it obtained the best classification results), \(\text{problem} - \text{VARCHAR}(10)\) (this is the classification problem).

The number of writing operations will be reduced as much as possible by making the inserts into the result table only when we have a better classification result (a lower PCICtest). This is the main optimization that was made in this case.

In my experiments I chose the "winner takes it all" policy for the combiner. Other combiner policies can be used also: ensemble averaging, boosting or mixture of experts.

The multilayer perceptron (MLP) was trained with the backpropagation algorithm.

The training (TR) and testing data (TS) was arranged as follows:
The classification problems that were analyzed are the satlog data set (classifying the type of terrain of an satellite image) and the heart data set (detection of a heart disease based on a number of input attributes):

<table>
<thead>
<tr>
<th>Table 1. Satlog data set</th>
</tr>
</thead>
<tbody>
<tr>
<td>satlog</td>
</tr>
<tr>
<td>Lines</td>
</tr>
<tr>
<td>Columns</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2. Heart data set</th>
</tr>
</thead>
<tbody>
<tr>
<td>heart</td>
</tr>
<tr>
<td>Lines</td>
</tr>
<tr>
<td>Columns</td>
</tr>
</tbody>
</table>

I also analyzed how performance is affected by the database storage engines that were used: MySql - MyISAM and InnoDB. Another subject to discuss based on the experiments is the type of replication that I had in the experiments: Statement based replication or Row based replication(Schwartz B., et al, 2008).

The benefit of working with more than one neural networks in terms of execution times is shown with the $T_d$ parameter which represents the most detrimental case (it will take the value of the execution of the slowest distributed system). I measured also the sequential run of the same neural networks: $T_s$. Based on those parameters I calculated the distributed speedup $S$.

\[
S_d = \frac{T_s}{T_d},
\]

\[
T_d = \max\{t_1, t_2, \ldots, t_n\}.
\]

For all the experiments I had five distributed system: four of them (P1, P2, P3, P4) were used to achieve the classification DM task and the last one was used as a combiner.

The first experiment was to see what execution performance I had for 4 neural network that work in a sequential way (the slowest system from the structure) for the two classification problems. MySql was on default settings(Figure 3, Figure 4).The DCM structure was not optimized in any way (all tables were replicated on all the systems - including the results one).

![Percentage of incorrect classification at testing](image-url)

**Figure 3.** Sequential execution for the satlog classification problem
Each multilayer perceptron run 10000 training epochs therefore I had 40000 training epochs for all neural networks.

The sequential performances are:
Satlog classification problem: \( T_s = 25962578 \text{ ms} \) (approx. 7 hours and 20 minutes).
Heart classification problem: \( T_s = 2773938 \text{ ms} \) (approx. 46 minutes)
We can see that the analyzed problems had bad performances in the sequential run.

3 Distributed execution with the MyISAM storage engine

The distributed execution on four distributed systems is presented in Figure 5 and Figure 6.
Figure 6. Distributed execution for the heart classification problem

All four distributed systems run 10000 training epochs in the same time. After every training epoch all MLPs run a testing epoch to initialize the PCICtest(or PCICg) value. The DCM structure was not optimized in any way regarding the design (all the insert operation are made on the server system and than replicated on all slave systems). The only optimization was to minimize the writing operation by setting the system to write in the distributed database only if we have a better classification result (PCICtest).

The performance of distributed execution for the Statement based replication is:

Table 3. Distributed performance - Statement replication

<table>
<thead>
<tr>
<th></th>
<th>Satlog - 36 neurons on the hidden layer</th>
<th>Heart - 35 neurons on the hidden layer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1 [ms]</td>
<td>T2 [ms]</td>
</tr>
<tr>
<td>Satlog</td>
<td>6527375</td>
<td>6497015</td>
</tr>
<tr>
<td>Heart</td>
<td>6654281</td>
<td>6631109</td>
</tr>
</tbody>
</table>

Satlog classification problem performance:
Distributed time Td = T1 = 6527375 ms (approx. 108 minutes).
Distributed speedup S = 3.97.

Heart classification problem performance:
Distributed time Td = T2 = 694546 ms (approx. 11 minutes).
Distributed speedup S = 3.99.

The performance of distributed execution for the Statement based replication is:

Table 4. Distributed performance - Row replication

<table>
<thead>
<tr>
<th></th>
<th>Satlog - 36 neurons on the hidden layer</th>
<th>Heart - 35 neurons on the hidden layer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1 [ms]</td>
<td>T2 [ms]</td>
</tr>
<tr>
<td>Satlog</td>
<td>6654281</td>
<td>6631109</td>
</tr>
<tr>
<td>Heart</td>
<td>6654281</td>
<td>6631109</td>
</tr>
</tbody>
</table>

Satlog classification problem performance:
Distributed time Td = T1 = 6654281 ms (approx. 108 minutes).
Distributed speedup S = 3.90.
Heart classification problem performance:
Distributed time $T_d = T_2 = 712735$ ms (approx. 11 minutes).
Distributed speedup $S = 3.89$.
When I used the optimized version of the DCM (Figure 1) I obtained the following results:

<table>
<thead>
<tr>
<th></th>
<th>Satlog - 36 neurons on the hidden layer</th>
<th>Heart - 35 neurons on the hidden layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$ [ms]</td>
<td>6315484</td>
<td>6616953</td>
</tr>
<tr>
<td>$T_2$ [ms]</td>
<td>6409000</td>
<td>6576406</td>
</tr>
<tr>
<td>$T_3$ [ms]</td>
<td>6232532</td>
<td>6303297</td>
</tr>
<tr>
<td>$T_4$ [ms]</td>
<td>6235703</td>
<td>6344496</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Satlog classification problem performance:
Distributed time $T_d = T_2 = 6409000$ ms.
Distributed speedup $S = 4$.
Heart classification problem performance:
Distributed time $T_d = T_2 = 689656$ ms.
Distributed speedup $S = 4$.
In the experimental test I counted for each perceptron (P1, P2, P3, P4) the number of operations for updating the data in the database:
P1 has conducted 16 operations update data.
P2 has conducted 15 operations update data.
P3 has conducted 23 operations update data.
P4 conducted eight operations update data.

4 Distributed execution with the InnoDB storage engine
Bellow I will present the final experimental results in case that we are using the InnoDB storage engine for the result table. The charts for these executions are the same as Figure 5 and Figure 6.

<table>
<thead>
<tr>
<th></th>
<th>Satlog - 36 neurons on the hidden layer</th>
<th>Heart - 35 neurons on the hidden layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$ [ms]</td>
<td>6616953</td>
<td>699547</td>
</tr>
<tr>
<td>$T_2$ [ms]</td>
<td>6576406</td>
<td>696875</td>
</tr>
<tr>
<td>$T_3$ [ms]</td>
<td>6303297</td>
<td>657797</td>
</tr>
<tr>
<td>$T_4$ [ms]</td>
<td>6344496</td>
<td>656734</td>
</tr>
</tbody>
</table>

Distributed time $T_d = T_1 = 6616953$ ms.
Distributed speedup $S = 3.95$.
Heart classification problem performance:
Distributed time $T_d = T_1 = 699547$ ms.
Distributed speedup $S = 4$.
The best classification results are: $PCIC_{test} = 16.59\%$ for the satlog problem and $PCIC_{test} = 21.5\%$ for the heart problem.
One important observation is that if we use more distributed neural network, we will have more diversity and the classification results should be better.

5 Conclusions
With these experiments I proved that it is very important to know how to design a DCM architecture for real implementations. A sequential execution of the backpropagation algorithm for a multitude of neural networks is time consuming(for the satlog classification problem it took 7 hours and 20 minutes to finish the training and testing process).
The distributed execution is much better than the sequential one (I obtained distributed speedup values very close to the number of distributed systems that I had - 4) and that the final result depends on the slowest system from the whole architecture.

I also showed that if we still want to have all the advantages of distributed databases and the best recovery mechanism of data we can still use the InnoDB storage engine from MySql with the condition of minimizing the number of the insert operations on the system that contains the combiner (the master server). In cases where many writings operations will take place is better to work with the optimized version of DCM., with the MyISAM storage engine because InnoDB uses compression of data (Rădescu, R., 2010; Rădescu, R., 2010), it has time consuming concurrency control and recovery methods for data (Schwartz B., et al, 2008).

In this paper I showed also that even the type of replication can affect the overall experiment. From the experimental results I can say that for this kind of application, the Statement based replication (Schwartz B., et al, 2008) is indicated to work with.

We can see that the study of interactions between the KDD process and real implementations of distributed databases can bring many improvements in designing DM structures.

This research is beneficial for fields like machine learning, distributed learning (Rădescu, R., Birkan, I., 2015; Rădescu, R., Soare, B., 2014; Rădescu, R., Davidescu, A., 2010), adaptive learning, biology, astronomy, medical research, financial research, medical diagnosis, gaming, management.

References


