# Scaling up classification rule induction through parallel processing

Presented by Melissa Kremer and Pierre Duez

## Background/Motivation

- Large datasets require parallel approaches
- Datasets can be:
  - Extremely large (NASA's satellites and probes send ~1TB of data per day)
  - Distributed (multiple sites)
  - Heterogeneous (multiple stakeholders with slightly different databases)

# Distributed Data Mining (DDM)

- DDM may refer to:
  - *Geographically* distributed data mining
    - Multiple data sources
    - Collaboration between multiple stakeholders
    - Cost/logistics to collating all data in one location
  - *Computationally* distributed data mining
    - Also referred to as “parallel data mining”
    - Scaling of data mining by distributing the load to multiple computers
    - Borne of a single, coherent dataset
- This paper focuses on the 2nd definition

# Outline

- Key concepts: Multiprocessor architectures, parallel data mining
- Data reduction
- Parallelizing loosely-coupled architectures
- Parallel formulations of classification rule induction algorithms
- Parallelization using PRISM
- Summary

# Multiprocessor Architectures and Parallel Data Mining

## Multiprocessor Architectures

Two types of multiprocessor architectures: tightly-coupled and loosely-coupled

- Tightly-coupled: processors use shared memory
- Loosely-coupled: each processor has its own memory

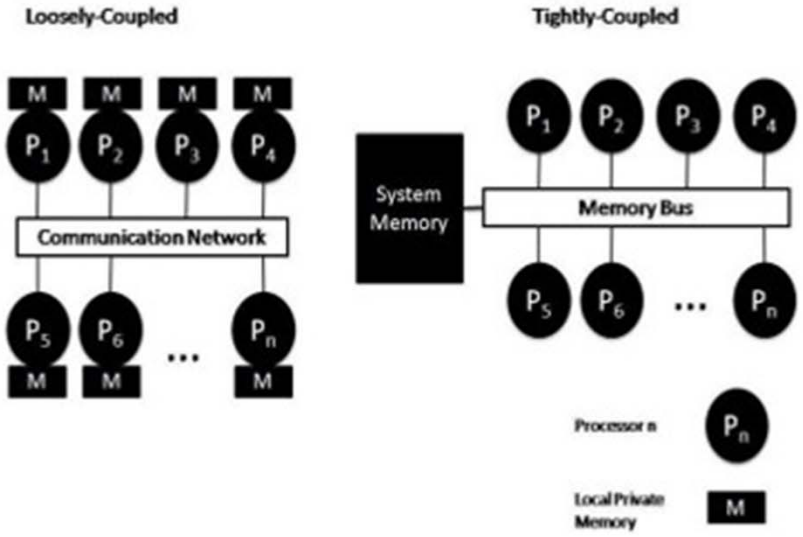



**Figure 1** Tightly- and loosely-coupled multiprocessor computer architectures

## Pros and Cons of Tightly-Coupled Systems

Tightly-coupled:

- Communication via memory bus system
- As number of processors increases, bandwidth decreases
- More efficient, avoiding data replication and transfer
- Costly to scale or upgrade hardware

## Pros and Cons of Loosely-Coupled Systems

Loosely-coupled:

- Requires communication between components over a network, increasing overhead
- Resistant to failures
- Easier to scale
- Components tend to be upgraded over time

## Data Parallelism vs. Task Parallelism

- Data Parallelism: same operations are performed on different subsets of same data.
- Task Parallelism: different operations are performed on the same or different data.

# Data Reduction

## Data Reduction Techniques

- Feature Selection
- Sampling

## Feature Selection

- *Information Gain* is a commonly-used metric to evaluate attributes
- General idea:
  - Calculate information gain
  - Prune features with lowest information gain
  - Calculate rules in reduced attribute space

## Feature Selection - stop conditions

- In step 2 (pruning attributes), stop conditions can include:
  - By number of attributes:
    - Keep *n* top attributes
    - Keep *x%* of the attributes with highest gain
  - By information gain
    - Keep all attributes whose information again is at least *x%* of the best attribute
    - Keep all attributes whose information gain is at least *x%*

## Feature Selection - Iterating

How to choose the best attributes?

From Han and Kamber (2001):

- Stepwise forward selection
- Stepwise backward elimination
- Forward and backward elimination
- Decision tree induction**

## or…

… just do PCA.

## Sampling

- Using a random sample to represent the entire dataset
- Generally, 2 approaches:
  - SRSWOR (Simple Random Sample WithOut Replacement)
  - SRSWR (Simple Random Sample With Replacement)
- Big design questions:
  - How to choose sample size?
  - OR How to tell when your sample is sufficiently representative?

## Sampling - 3 techniques

- Windowing (Quinlan, 1983)
- Integrative Windowing (Fuernkranz, 1998)
- Progressive Sampling (Provost et al., 1999)

## Sampling - Windowing

- The algorithm:
  - Start with user-specified window size
  - Use sample (“window”) to train initial classifier
  - Apply classifier to remaining samples in dataset (until limit of misclassified examples is reached)
  - Add misclassified samples to the window and repeat
- Limitations:
  - Does not do well on noisy datasets
  - Multiple classification/training runs
  - Naive stop conditions

## Sampling - Windowing (cont’d)

Extensions to Windowing (Quinlan, 1993) including:

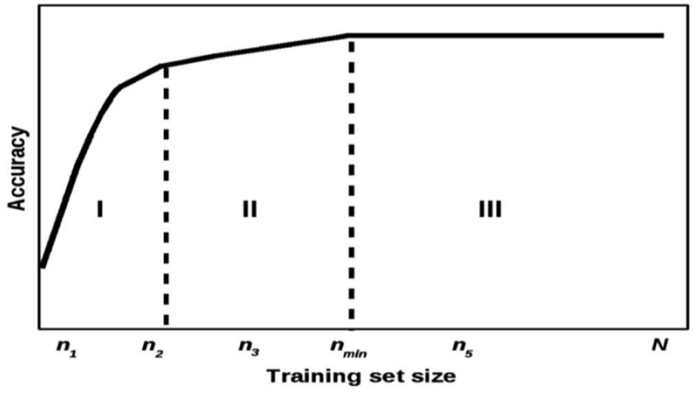
- Stopping when performance stops improving
- Aim for uniformly distributed window, which can help accuracy on skewed datasets

# Integrative Windowing

- Extension of Quinlan (1983)
- In addition to adding misclassified examples to window, deletes instances that are covered by consistent rules.
  - "Consistent Rule": rule that did not misclassify a negative example
- Consistent Rules are remembered, but get tested on future iterations (to ensure full consistency)

# Progressive Sampling

- Make use of relationship between sample size and model accuracy
- 3 Phases (ideally) of learning:
- Goal: find $n_{min}$, where the model accuracy plateaus




- Limitation: assumptions made about accuracy vs training set size

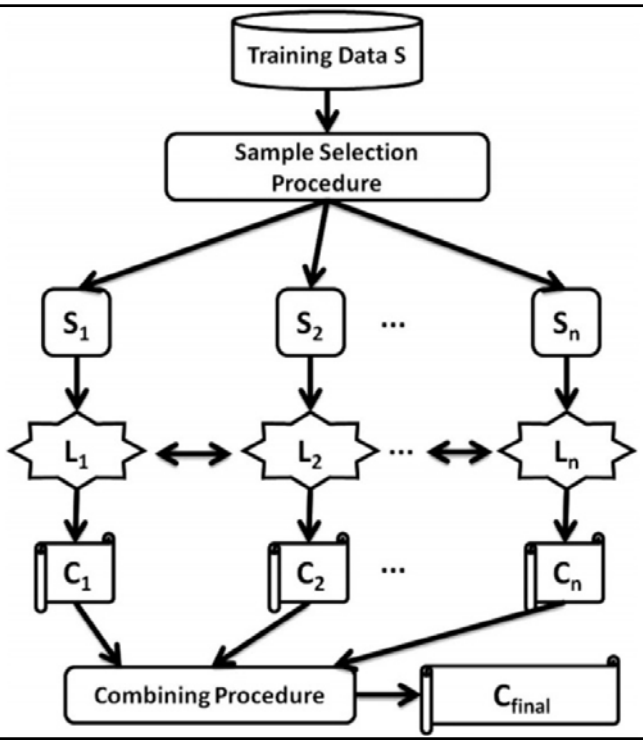
# Parallelizing Loosely-coupled Architectures

## Parallelizing

- In a loosely-coupled architecture, we partition the data into subsets and assign the subsets to n machines
- Want to distribute data so that workload is equal across machines

# Three Basic Steps of Parallelization

1. Sample selection procedure
   - The dataset may be divided into equal sizes, or sizes may reflect speed or memory of the individual machines
2. Learning local concepts
   - a learning algorithm is used to learn rules from local dataset
   - Algorithms may or may not communicate training data or information about the training data
3. Combining local concepts
   - Use a combining procedure to form a final concept description



- Invariant partitioning property:
  - every rule that is acceptable globally (according to a metric) must also be acceptable on at least one data partition on one of the n machines

# Parallel Formulations of Rule Induction Algorithms

## Top-Down vs Bottom Up

- Top-Down Induction of Decision Trees (TDIDT) generally follow “Divide & Conquer” approach:
  - Select attribute (A) to split on
  - Divide instances in the training set into subsets for each value of A
  - Recurse along each leaf until stop condition (pure set, exhausted attributes, no more information gain, etc.)
- Authors refer to alternative as “Separate & Conquer” (rule sets)
  - Repeatedly generate rules to cover as much of the outstanding training set as possible
  - Remove samples covered by the rules from the set, and train new rules on the remaining examples

## Two Main Approaches

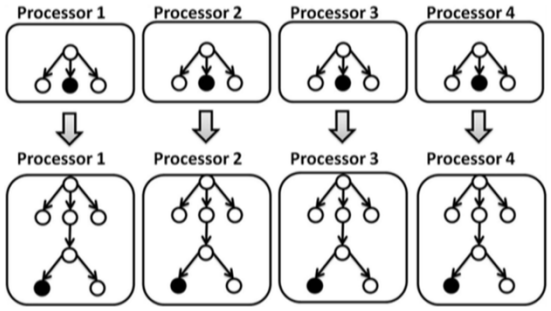
- Parallel formulations of decision trees
- Parallelization through ensemble learning

## Parallel Formulations of Decision Trees

- Synchronous Tree Construction
- Partitioned Tree Construction
- Vertical Partitioning of Training Instances

## Synchronous Tree Construction

- Loosely coupled system: each processor has its own memory, bus
- Processors work synchronously on the active node, report distribution characteristics and collectively determine next partition



## Synchronous Tree Construction

Advantages:

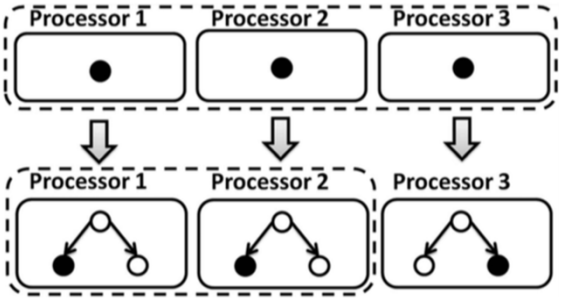
- No communication of training data required => fixed communication cost

Disadvantages

- Communication cost dominates as tree grows
- No way to balance workload

## Partitioned Tree Construction

- Processors begin as synchronized parallel tree construction
- As number of nodes increases, they are assigned to a single processor (along with descendents)



# Partitioned Tree Construction

Advantages

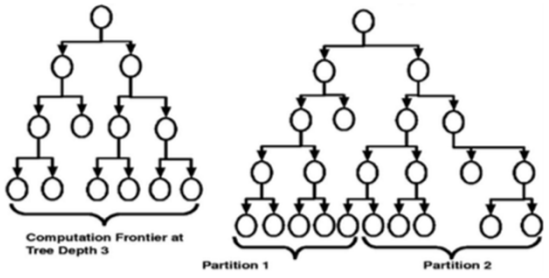
- No overhead for communication or data transfer at later levels
- Moderate load balancing in early stages

Disadvantages

- First stage requires significant data transfer
- Workload can only be balanced on the basis of the number of instances in higher nodes; cannot be rebalanced if one processor finishes early

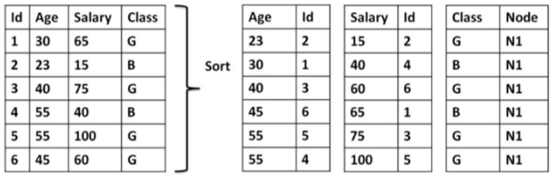
# Hybrid Tree Construction

- Sirvastava et al, 1999:
  - Begin with synchronous tree construction
  - When communication cost becomes too high, partition tree for later stages



# Vertical Partitioning of Training Instances

- SLIQ: Super Learning in Quest
- Split database into *<attribute value, class index>* pairs (as well as a *<class-label, node>* list for the classes.
  - Sorting for each attribute only has to happen once
- Only need to load one attribute-projected table (and the class/node table) at a time
- Splitting is performed by updating the *<class-label, node>* table.



| Id | Age | Salary | Class |
|---|---|---|---|
| 1 | 30 | 65 | G |
| 2 | 23 | 15 | B |
| 3 | 40 | 75 | G |
| 4 | 55 | 40 | B |
| 5 | 55 | 100 | G |
| 6 | 45 | 60 | G |

Sort

| Age | Id |
|---|---|
| 23 | 2 |
| 30 | 1 |
| 40 | 3 |
| 45 | 6 |
| 55 | 5 |
| 55 | 4 |

| Salary | Id |
|---|---|
| 15 | 2 |
| 40 | 4 |
| 60 | 6 |
| 65 | 1 |
| 75 | 3 |
| 100 | 5 |

| Class | Node |
|---|---|
| G | N1 |
| B | N1 |
| G | N1 |
| B | N1 |
| G | N1 |
| G | N1 |

# SLIQ - disadvantages

- Memory footprint: still need to have class, attribute tables for all records in memory at once
- Tricky to parallelize:
  - SLIQ-R (replicate class list)
  - SLIQ-D (distributed class list)
  - Scalable Parallelizable Induction of Decision Trees (SPRINT)

# SPRINT (Shafer et al., 1996)

- Similar to SLIQ
- Builds Attribute-ID-Class tuples
- Nodes information is encapsulated in separation into sub-lists

SLIQ :

| Id | Age | Salary | Class |
|---|---|---|---|
| 1 | 30 | 65 | G |
| 2 | 23 | 15 | B |
| 3 | 40 | 75 | G |
| 4 | 55 | 40 | B |
| 5 | 55 | 100 | G |
| 6 | 45 | 60 | G |

Sort

| Age | Id |
|---|---|
| 23 | 2 |
| 30 | 1 |
| 40 | 3 |
| 45 | 6 |
| 55 | 5 |
| 55 | 4 |

| Salary | Id |
|---|---|
| 15 | 2 |
| 40 | 4 |
| 60 | 6 |
| 65 | 1 |
| 75 | 3 |
| 100 | 5 |

| Class | Node |
|---|---|
| G | N1 |
| B | N1 |
| G | N1 |
| B | N1 |
| G | N1 |
| G | N1 |

SPRINT:

| Age | Salary | Class |
|---|---|---|
| 30 | 65 | G |
| 23 | 15 | B |
| 40 | 75 | G |
| 55 | 40 | B |
| 55 | 100 | G |
| 45 | 60 | G |

Sort

N1 (Root node)

| Age | Id | Class |
|---|---|---|
| 23 | 2 | B |
| 30 | 1 | G |
| 40 | 3 | G |
| 45 | 6 | G |
| 55 | 5 | G |
| 55 | 4 | B |

| Salary | Id | Class |
|---|---|---|
| 15 | 2 | B |
| 40 | 4 | B |
| 60 | 6 | G |
| 65 | 1 | G |
| 75 | 3 | G |
| 100 | 5 | G |

N2 (Salary < 60)

| Age | Id | Class |
|---|---|---|
| 23 | 2 | B |
| 55 | 4 | B |

| Salary | Id | Class |
|---|---|---|
| 15 | 2 | B |
| 40 | 4 | B |

N3 (Salary ≥ 60)

| Age | Id | Class |
|---|---|---|
| 30 | 1 | G |
| 40 | 3 | G |
| 45 | 6 | G |
| 55 | 5 | G |

| Salary | Id | Class |
|---|---|---|
| 60 | 6 | G |
| 65 | 1 | G |
| 75 | 3 | G |
| 100 | 5 | G |

# SPRINT - multiple processors

- To parallelize SPRINT, divide the list up into multiple sub-lists
- Processors then calculate local split-points
- Need to track globally best split-point, using hash table
- Limitation: hash table needs to be shared, and scales up with number of records

## Parallelization through Ensemble Learning

- Concept:
  - Partition the data into manageable subsamples
  - Subsamples sized to fit individual machines/processor memories
  - Create collection of independently-trained classifiers
  - Combine* to create final classifier

## Ensemble Learning (cont'd)

Advantages:

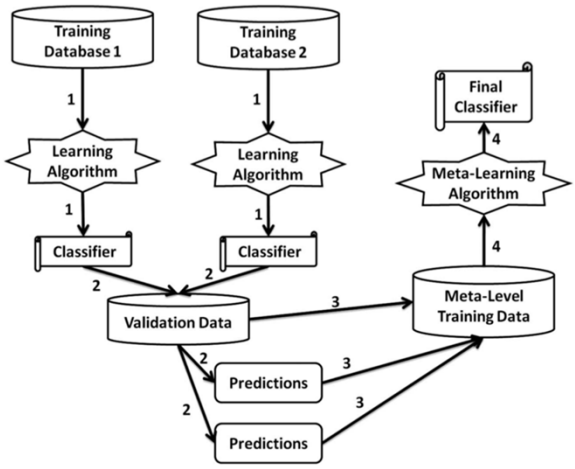
- No communication necessary between training batches = faster training time

Disadvantages:

- Less accurate than a single classifier trained on entire dataset

# Refinement: Meta-classifier

- Train base classifiers as before
- Train meta-classifier on outputs of classifiers on new data
- Possible meta-strategies:
  - *Voting* (majority or weighted)
  - *Arbitration* (if consensus is not reached)
  - *Combining* (rule set for classifiers)




# Parallelization Using PRISM

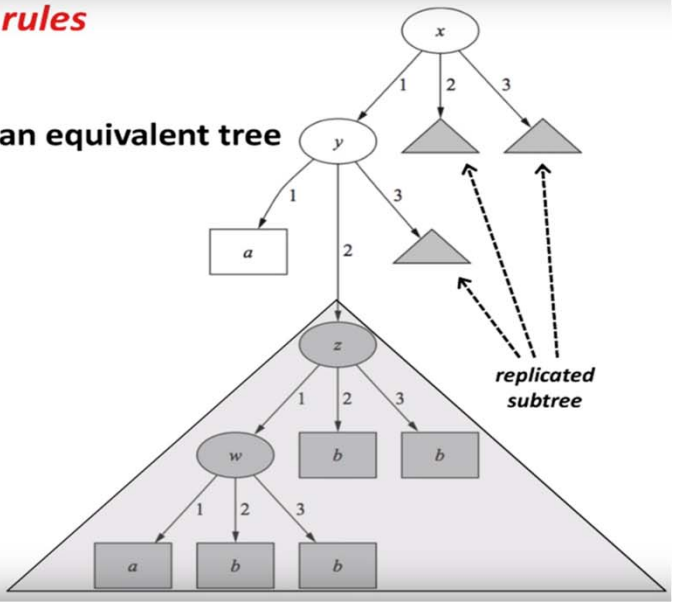
# Why Learn Rules?

***Lesson 3.1: Decision trees and rules***

**For any set of rules there is an equivalent tree**

*but it might be very complex*

if x = 1 and y = 1 then a
if z = 1 and w = 1 then a
otherwise b

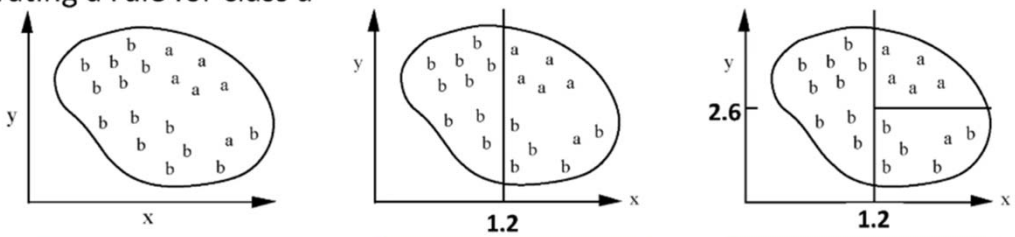



# Simple bottom-up covering algorithm for creating rules: PRISM

For each class *C*
  Initialize *E* to the instance set
  While *E* contains instances in class *C*
    Create a rule *R* that predicts class *C*
      (with empty left-hand side)
    Until *R* is perfect
      (or there are no more attributes to use)
      For each attribute *A* not mentioned in *R*, and each value *v*
        Consider adding the condition *A* = *v* to the left-hand side of *R*
        Select *A* and *v* to maximize the accuracy
          (break ties by choosing the condition with the largest *p*)
      Add *A* = *v* to *R*
    Remove the instances covered by *R* from *E*

Images taken from: https://www.youtube.com/watch?v=N_V2BmjeYLw

Generating a rule for class *a*



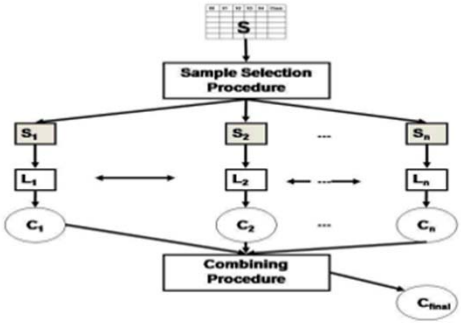

if *true*
then class = *a*

if x > 1.2
then class = *a*

if x > 1.2 and y > 2.6
then class = *a*

- Rules are chosen based on Information Gain comparison, discussed in class
- PRISM only works with discrete values. Continuous values must be discretized
- There may be issues if the training data contains a clash set.
  - E.g. instances have the same values for attributes but are assigned to different classes
  - These can be dealt with by discarding the rule if the target class is not the majority class, then deleting the instances that are assigned to the discarded rule's target class

# PMCRI for Parallelizing Prism




- Based on the Cooperative Data Mining (CDM) model
- 3 steps: sample selection, local learning, combination into final concept description

# Dividing the Workload

- Build attribute lists similar to SPRINT, then distribute them evenly over p processors
- However, attribute lists are not further divided into p parts. The attribute lists are distributed whole. This prevents imbalances later on
- So every node is working with the same instances but local rules are generated for different attributes


**Figure 15** The left hand side shows how sorted attribute lists are built and the right hand side shows how list records, in this case records with the ids 1 and 3, are removed in Prism, after a rule term has been found.

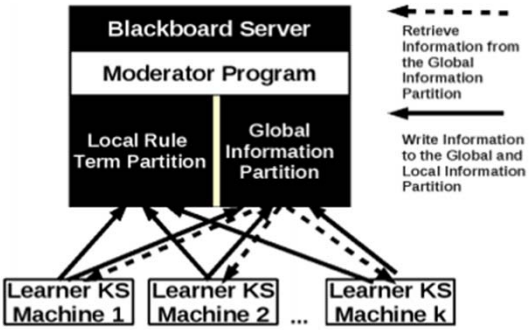



**Figure 16** PMCRI's communication pattern using a distributed blackboard architecture.

- Step 1: Moderator writes on Global Information Partition the command to induce locally best rule terms.
- Step 2: All KSs induce the locally best rule term and write the rule terms plus its covering probability on the local Rule Term Partition
- Step 3: Moderator compares all rule terms written on the Local Rule Term Partition and writes the name of the KS that induced the best rule term on the "Global Information Partition"

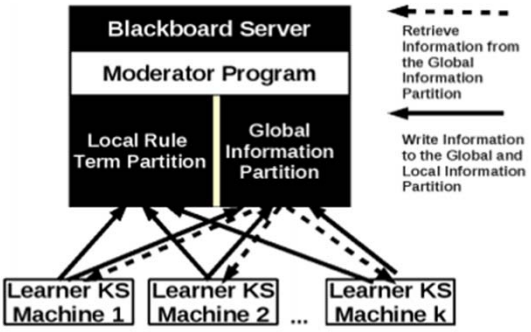



**Figure 16** PMCRI's communication pattern using a distributed blackboard architecture.

- Step 4: KS retrieves name of winning expert.
  - IF KS is winning expert {
    - keep locally induced rule term and derive by last induced rule term uncovered ids and write them on the Global Information
    - Partition and delete uncovered list records

  }
  - ELSE IF KS is not winning KS {
    - delete the locally induced rule term and wait for by best rule term uncovered ids being available on the Global Information Partition, download them
    - delete list records matching the retrieved ids.

  }

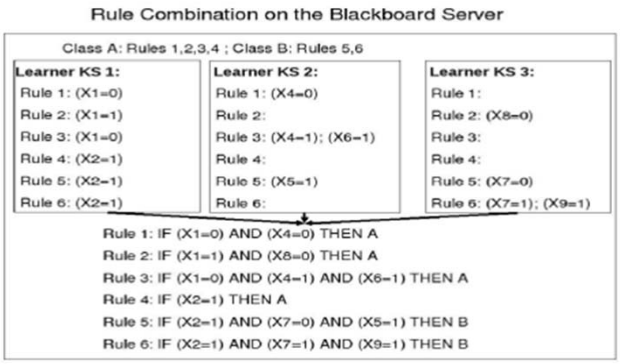



**Figure 17** Combining Procedure.

- Each learner KS builds rule terms for the same rule simultaneously in a part-rule list
- The rules are then evaluated globally and the best rules are chosen
- The PMCRI framework is able to reproduce exactly the same result as any serial Prism algorithm
- Communication to build global rules is only done at the end, reducing overhead

# J-PMCRI and Evaluations

- A method of pre-pruning for PMCRI, called J-PMCRI, was developed and discussed in another paper
- Runtime of a fixed processor configuration (learner KS machine) on an increasing workload was examined
  - These are called size up experiments in contrast with speed up experiments that keep the workload constant and increase the number of processors.

## Results

- For PMCRI and J-PMCRI the workload is equivalent to the number of data records and attributes that are used to train a Prism classifier
- The runtime can be optimized as a linear function of the data set size. Memory consumption is also linear with respect to the size of the training data set
- The larger the amount of data used, the more PMCRI and J-PMCRI benefit from using additional processors

## Strengths and Limitations

Strengths:

- Allows faster runtime through parallelization
- Gives the same results as PRISM
- Runtime, memory consumption, etc. is linear with respect to the size of the datasets
- Resistant to failures

Weaknesses:

- Loosely-coupled architecture, so same drawbacks
- Communication is required, but only in the combination step
- Pre-pruning the dataset is more difficult because the pre-pruning must also be parallelised. Explored in other papers

# Summary

## One-Slide Recap

- Big data (genomics, chemistry, astronomy, business, etc.) are producing datasets that cannot be held in memory
- Increased need for DM approaches that scale to multiple computers (more processors & more memory, but slower communication between processors)
- Tightly coupled vs Loosely coupled systems for Distributed Data Mining
- Data parallelism vs Task parallelism
- Data reduction techniques
- Distributed Data Mining Models: parallelizing loosely-coupled architectures
- Parallelization of rule learning: synchronous tree construction, partitioned tree construction, ensemble learning
- Speeding up “Separate and Conquer” approaches: parallelizing PRISM with PMCRI

# Limitations

- Age of this paper (published in 2012): authors *predict* increased use of multiprocessor systems:

  > "Nowadays, *dual-core* processors are standard technology, but in the near future we will have *multi-core* processors in our personal computers." (emphasis in the original)

- Authors concede that more systems will be tightly-coupled; point out that many of these lessons will continue to apply.