Knowledge Discovery in Databases II

Winter Term 2014/2015

Lecture 6:
Volume: Large Object Cardinalities: Parallel and Distributed Data Mining

Lectures : Dr Eirini Ntoutsi, PD Dr Matthias Schubert
Tutorials: PD Dr Matthias Schubert

http://www.dbs.ifi.lmu.de/cms/Knowledge_Discovery_in_Databases_II_(KDD_II)

Chapter Overview

1. Solutions for Large Object Cardinalities
2. Parallel and Distributed Data Mining
3. Privacy Preserving Data Mining
4. Sampling and Summarization

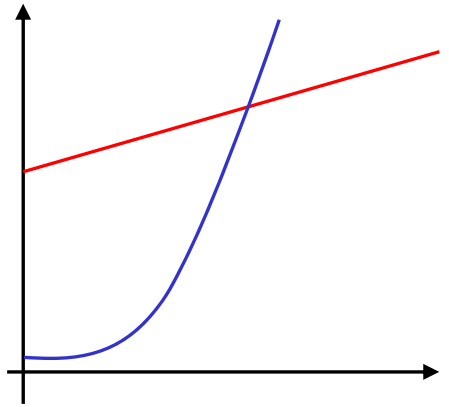
Problem Setting

So far:

- Focus on quality: How can we derive meaningful patterns?
- Data mining tasks yield high complexities

In this chapter:

- How can we mine high volumes of data faster?
- Performance depends on
  - the volume of the data set (# records/instances)
  - the scalability of the data mining algorithms



Solutions for Speeding up Data Mining

1. Use modern hardware
   - Parallel Data Mining
   - Distributed Data Mining
   - Privacy Preserving Data Mining
2. Reduce the number of objects being processed
   - Sampling
   - Summarization

Speeding up through Modern Hardware 1/2

Use modern hardware to speed up data mining:

- Cloud Computing => Parallel Data Mining
- Broadband Networks => Distributed Data Mining

Where does it help?

- high volume data repositories (electronic payments, sales data, web pages, emails, ...)

  => every data object must be examined at least once
- preprocessing (select relevant data objects)
- data transformation (data discretization, temporal aggregation etc.)

Speeding up through Modern Hardware 2/2

Limitations of high-performance computing architectures:

- best-case speed up of parallel algorithms: linear in the number of machines
- in most cases: less than linear due to communication and result merging overheads
- in problems having a super linear complexity: adding more machines helps but does not make the problem scalable

What can be done in these cases?

Speeding up through Sampling and Summarization 1/2

Reduce the number of objects being processed

$\Rightarrow$ Sampling and Summarization

Why does this make sense ?

Representative Sample $\neq$ Large Sample

- A too small data set might not be representative
- A very large data set can still be biased and not representative

=> there are redundant samples

=> removing these from the data set does not hurt the representativeness

Speeding up through Sampling and Summarization 2/2

Methods for reducing large data sets:

- Sampling:
  - Use a subset of the data set by removing redundant instances
  - Find redundant features
- Summarization
  - Instead of raw data records, use their summaries
- A popular summary: Microclusters:
  - use a clustering algorithm to determine a set of cluster descriptions
  - perform data mining on cluster descriptions

Parallel and Distributed Data Mining

Goal:

- use multiple cores /work stations to increase performance
  $\Rightarrow$ parallel data mining

- if data is stored in distributed locations:
  $\Rightarrow$ distributed data mining

- if data is confidential:
  $\Rightarrow$ privacy preserving data mining

  Privacy can only be preached if there are at least two parties (data owner and data user).

  => closely related to distributed mining

Examples

- Clustering end customers for distributors:
  - Retailer do not want to share customer information but might share distributions or statistics
  - Retailer needs *„privacy-preserving“* Clustering algorithms to derive general end customer groups

- Pharmaceutical companies collect costumer sales data from pharmacies
  - helps the company to plan the production of pharmaceutics
  - find profitable areas for researching new drugs

  But: Individual drug consume of costumers might be sold to insurance companies or is made available to the public.
  (potential employers, landlords, credit institutions,..)

Parallel Data Mining

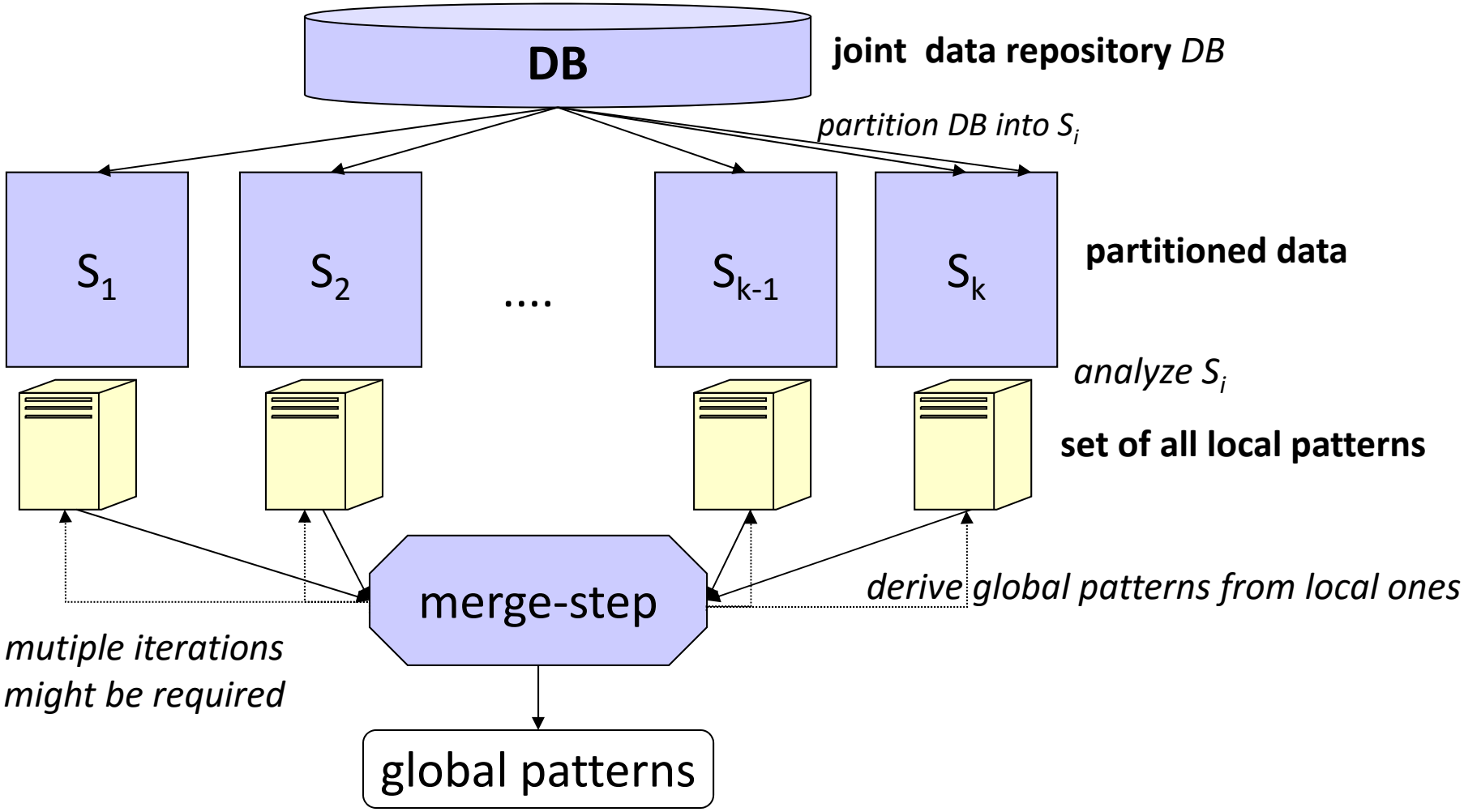
Parallel Data Mining:

- Data repository is already integrated and available in a common location.
- data has to be analyzed on *k* work stations
- performance gain by following a "*Divide and Conquer*" strategy:
  - $\Rightarrow$ distribute data to worker tasks
  - $\Rightarrow$ each worker analyses the data and returns a local result
  - $\Rightarrow$ local results must be combined to global patterns/functions

Important aspects to consider:

- Distribute data in a way that joining local results into global patterns is easy
- Avoid communication between the workers as much as possible

Workflow of Parallel Data Mining



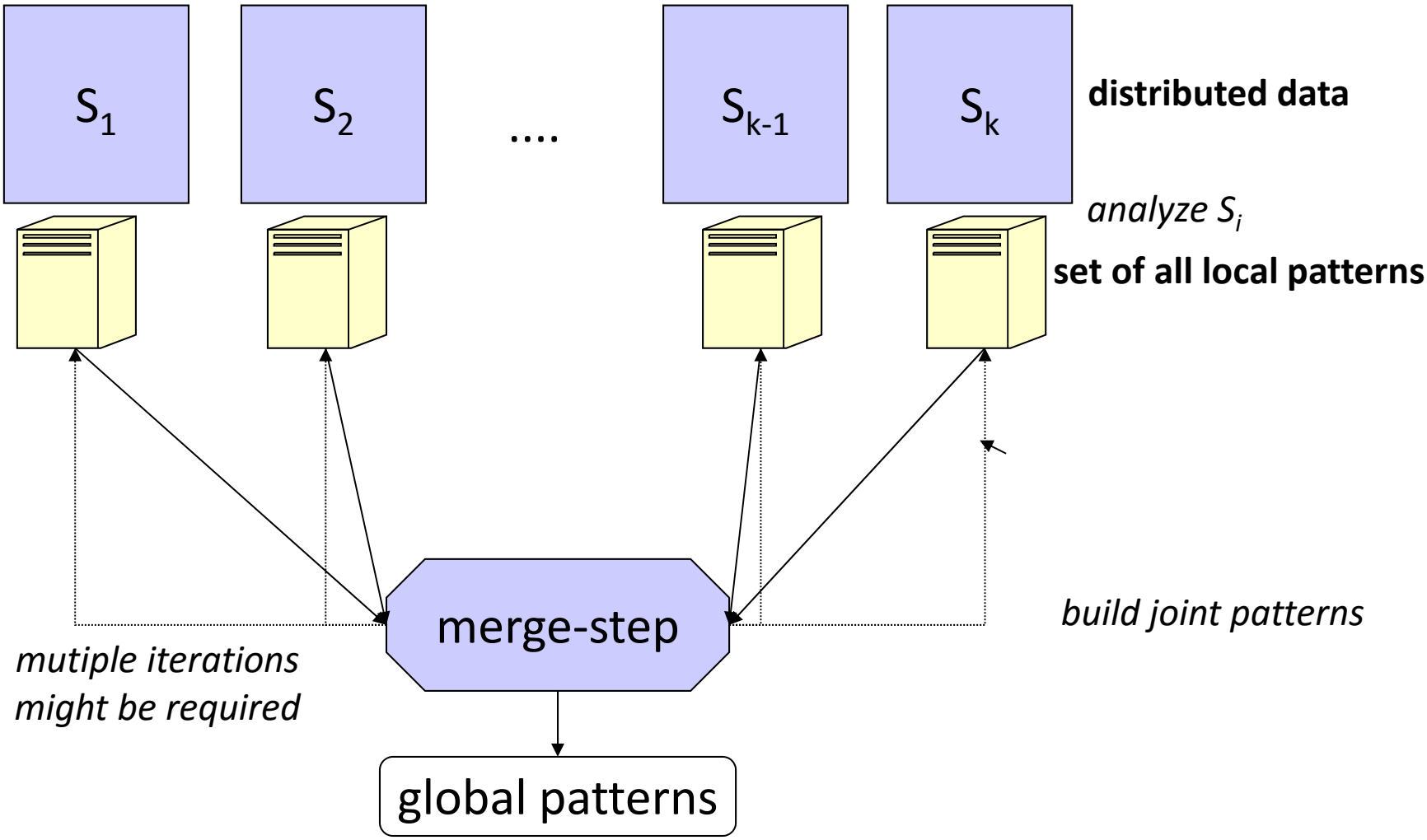
Distributed Data Mining

- The distribution of data to the different peers is given
  - ⇒ no effort for data partitioning
  - ⇒ local patterns are less controllable
  - ⇒ joining local patterns might be more difficult

- an unfavorable distribution might lead to the following problems:
  - Discrepancies between the result of distributed and stationary mining
  - Large communication effort

- Differences between parallel and distributed data mining:
  - distribution is given
  - network costs are usually assumed to be higher (between companies, mobile clients..)

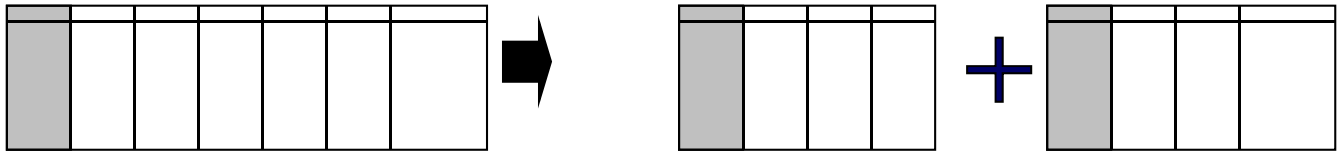
Workflow of Distributed Data Mining



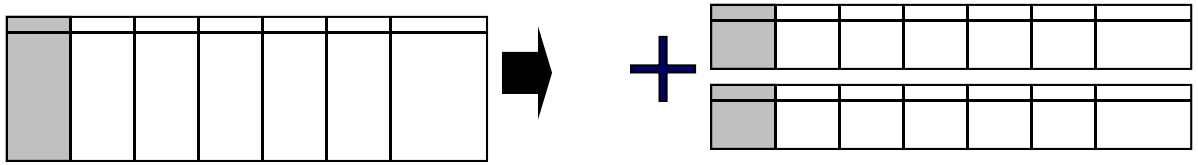
Important aspects: Data partitioning

- Data partitioning= physically dividing data in different data stores
  - Improves scalability, performance, availability, security

- Vertical Partitioning
  - Features are distributed. Objects are available everywhere

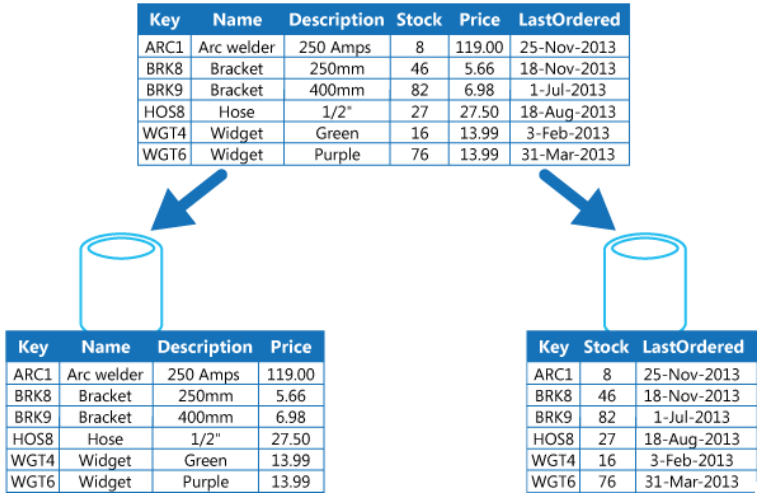


- Horizontal Partitioning
  - Objects are distributed over workers and sites. Object description is everywhere the same.



- In practice: Data might be partitioned in both ways.

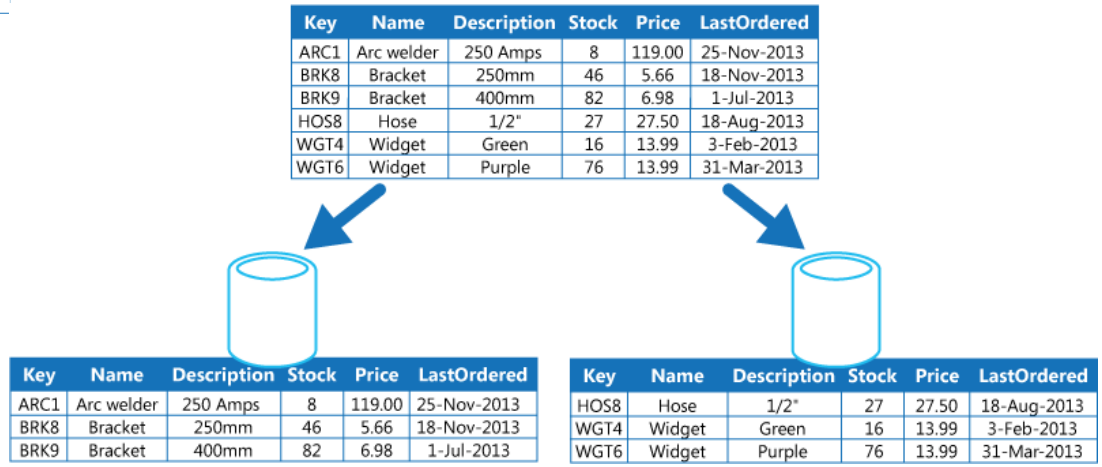
Data partitioning examples



Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Price
ARC1	Arc welder	250 Amps	119.00
BRK8	Bracket	250mm	5.66
BRK9	Bracket	400mm	6.98
HOS8	Hose	1/2"	27.50
WGT4	Widget	Green	13.99
WGT6	Widget	Purple	13.99

Key	Stock	LastOrdered
ARC1	8	25-Nov-2013
BRK8	46	18-Nov-2013
BRK9	82	1-Jul-2013
HOS8	27	18-Aug-2013
WGT4	16	3-Feb-2013
WGT6	76	31-Mar-2013



Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013

Key	Name	Description	Stock	Price	LastOrdered
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Source: https://msdn.microsoft.com/en-us/library/dn589795.aspx

Further Important Aspects

2. **Data Mining Task:** Classification, Clustering, Association Rules

3. **Partitioning dependency:** Does the result depend on the used/given partitioning of the data?

4. **Type of local patterns:** Approximations, data objects, distributions… Examples: Gaussians, hyper rectangles, centroids…

5. **Organization of the distributed workflow:**
   Master and slave processes, P2P computation

Chapter Overview

1. Solutions for Large Object Cardinalities
2. Parallel and Distributed Data Mining
3. Privacy Preserving Data Mining
4. Sampling and Summarization

Parallel Data Mining Algorithms

- Usually the result is expected to be independent from the partitioning (deterministic result)
- Main focus is speeding up the computation
- The partitioning strategy is often a major part of the algorithm:
  - minimize effort for joining local patterns

    ⇒ local patterns should be independent from each other

    ⇒ in case of dependencies: extra communication is required or inaccurate results have to be accepted

  - Runtime depends on the worst runtime of any worker task

    ⇒ all parallel steps should take about the same amount of time

    ⇒ all sites should receive the same amount of data

Parallelism via Database Primitives

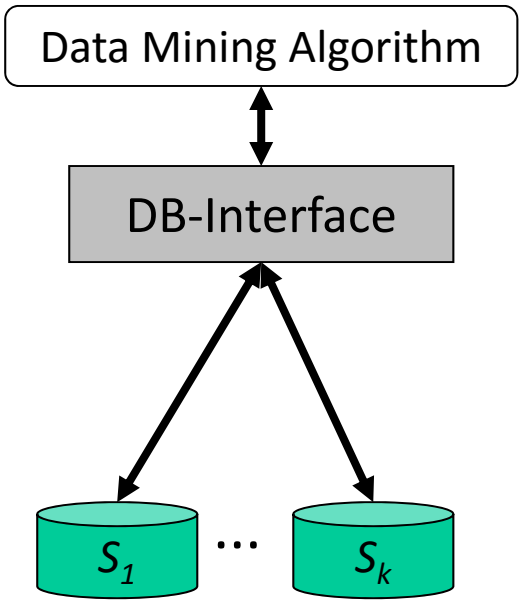
- In general data mining algorithms can be based on database primitives (e.g. ε-range queries, kNN queries).
- parallel computing of the database primitives yields a better support to general data mining algorithms.

Example:

- parallel computation of ε-range queries can accelerate density-based clustering
- parallel kNN queries allow fast kNN classification.

Characteristics:

- The join of the results has to be done on one machine
- Partitioning might still play a major role



distributed query processing

Parallel Density-based Clustering

Idea:

- Horizontal and compact partitioning
- Determine local core points and clusters
- Connect local clusters to global clusters:
  - Clusters from different sites
  - Noise points from other sites

General problem:

- What happens with objects where their ε-range intersects with other partitions?
  - mirror marginal objects
  - requires communication between the partitions

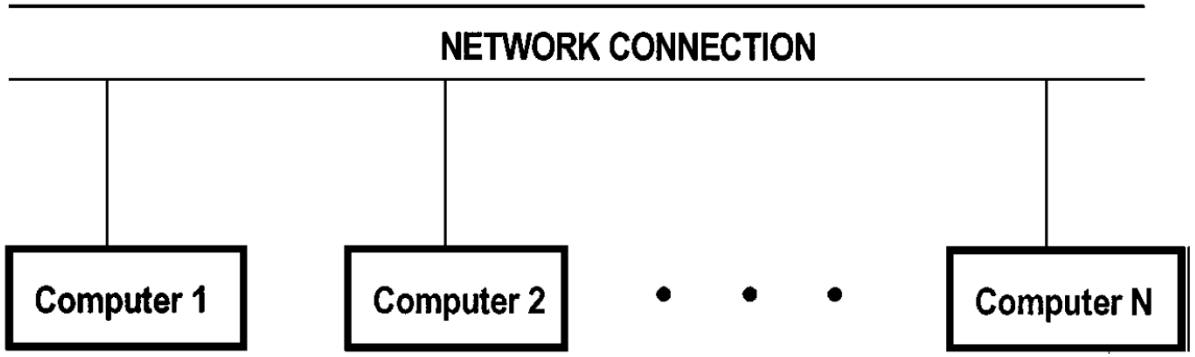
PDBSCAN [XuJaeKri99]

The problem:

Given a set of *d*-dimensional points $DB = \{p_1, p_2, \ldots p_n\}$, a minimal density of clusters defined by *Eps* and *MinPts*, and a set of computers $CP = \{C_1, C_2, \ldots C_N\}$ connected by a message passing network, find the density-based clusters with respect to the given *Eps* and *MinPts* values.

The hardware architecture:

Use a "shared-nothing" architecture with multiple computers interconnected through a network



PDBSCAN overview

- Three main steps

  [Step 1]: divide the input into several partitions, and distribute these partitions to the available computers.

  [Step 2]: cluster partitions concurrently using DBSCAN.

  [Step 3]: combine or merge the clusterings of the partitions into a clustering of the whole database.

- Pseudocode

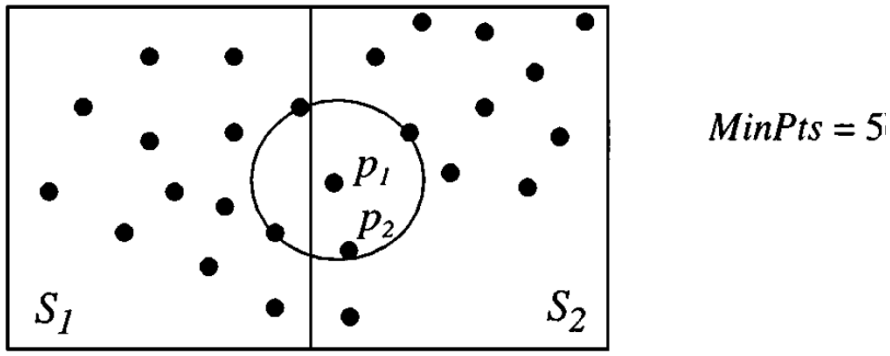
1. divide the input data set DB into N partitions $S_1, S_2, \ldots, S_N$ such that $DB = \cup_{i=1}^{N} S_i$ and $S_i \cap S_j = \emptyset$, for $i \neq j$. The partition S_i is distributed on C_i where $i = 1, 2, \ldots, N$.
2. process the N partitions concurrently using DBSCAN on the available computers C_1, $C_2, \ldots, C_N$., i.e. call algorithm DBSCAN(S_i, *Eps*, *MinPts*) concurrently on C_i for $i = 1, 2, \ldots, N$.
3. merge the clustering results obtained from the partitions $S_i, i = 1, 2, \ldots, N$, into a clustering result for DB.

Step 1: data placement

- Requirements for data placement
  - Load balancing: The data should be placed such that in step 2, all concurrent parallel DBSCAN(S_i, *Eps*, *MinPts*), *i=1:N*, will be finished at the same time.
    - Since the run-time of DBSCAN only depends on the size of the input data, the partitions should be almost of equal size if we assume that all computers have the same processing (computing and *I/O*) performance.
  - Minimized communication cost: The data should be placed such that the communication cost is minimized.
    - each local DBSCAN should avoid accessing data located on any of the other computers. Nearby objects should be organized on the same computer.
  - Distributed data access: The data should be placed such that both local and remote data can be efficiently accessed.
    - Locally DBSCAN needs $O(|S_i|^2)$, which can be improved through some index structure to $O(S_i \log(S_i))$.

Why we need access to remote data?

- If no support for accessing remote data, p_1 is not core in S_2 and p_2 is not density reachable by any point in S_2
- To obtain correct clustering, a "view" over the border of partitions is therefore necessary

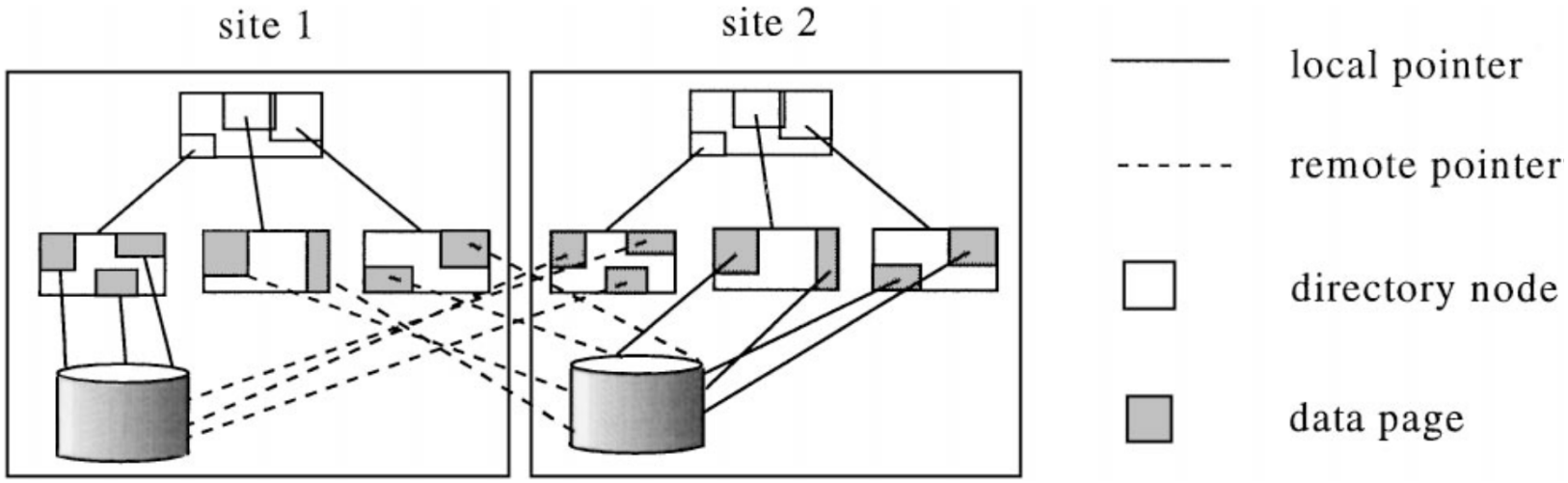


- But, we have to pay communication cost for every access to remote data.
  - It can be minimized by the replication of indices
  - It is only required for the objects located on the border of two neighboring partitions.
  - Another pay-off of remote data access is that we can efficiently merge the clustering results.

dR*-tree (distributed R*-tree)

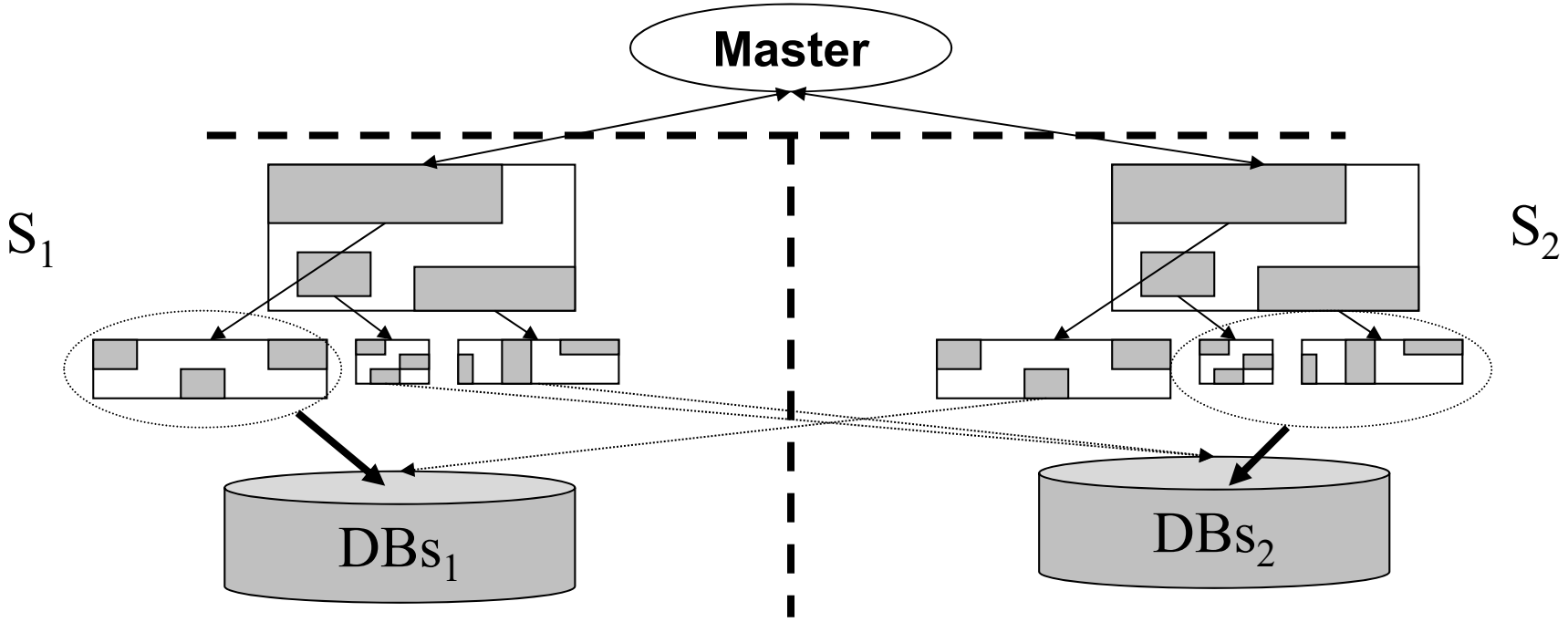
Idea: dR*-tree

- Group the MBRs (Minimum Bounding Rectangle) of the R*-tree into N partitions such that nearby MBRs are assigned to the same partition and partitions hold a similar number of MBRs.
- Distribute the partitions on all available computers
- Replicate the directory of R* on all available computers



The dR*-tree index

dR*-tree



- Provides efficient access to both local and remote data
- Queries on S_i, being completely processible on DBs_i can be answered completely simultaneously
- Access to pages on other sites reduce concurrence and raise communication costs

=> Algorithms should employ as much local queries as possible

Step 2: Local clustering PartDBSCAN

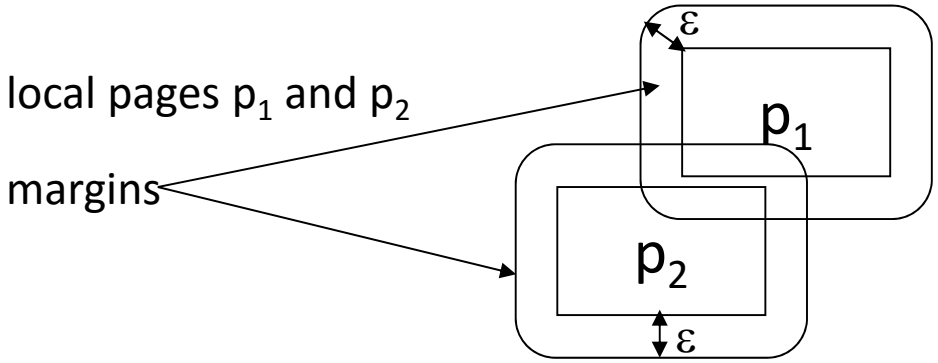
- Data partitioning in the dR*-tree
- Implemented using the master-slave model
- Slaves are responsible for local clustering and sending results to the master

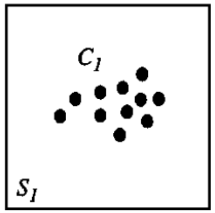
- PartDBSCAN(*S, dR*-tree, Eps, MinPts*) for local clustering in partition *S*
  - Modified DBSCAN handling only data within *S*
  - Starts with a point p in S and retrieves all points that are density reachable from p in the space constraint S.
    - If ε-range intersects with the margin:
      - Margins might have to be loaded from other sides to determine core point
      - Expanding clusters beyond the margin is to expensive would lead to large communication overheads.

      => store clusters with points outside S in merge list
  - A cluster C found in S is not necessarily a global cluster
    - If there are members of C outside S, C might need to be merged with another cluster found w.r.t. an adjacent space constraint.
    - C is called merging candidate and is sent to the master
  - At the end, merging candidates are sent to the master.
  - No need to send the whole (local) cluster, only points near the border of S.

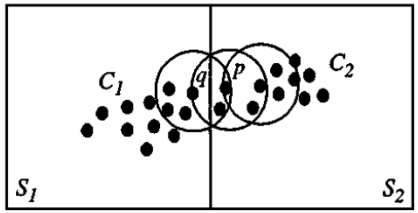
Step 3: Global Clustering

- Join local cluster having common merge points: merge point needs to be a core point in at least one partition => merge clusters

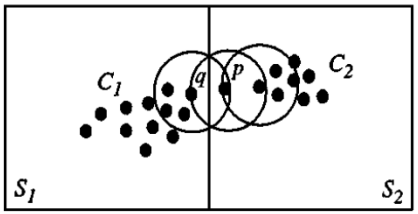




(a) $C_1 \subseteq S_1$, C_1 is a cluster found wrt. the space constraint *DB*



(b) $p \in C_1 \backslash S_1$ is a core point; C_1 should be merged with C_2



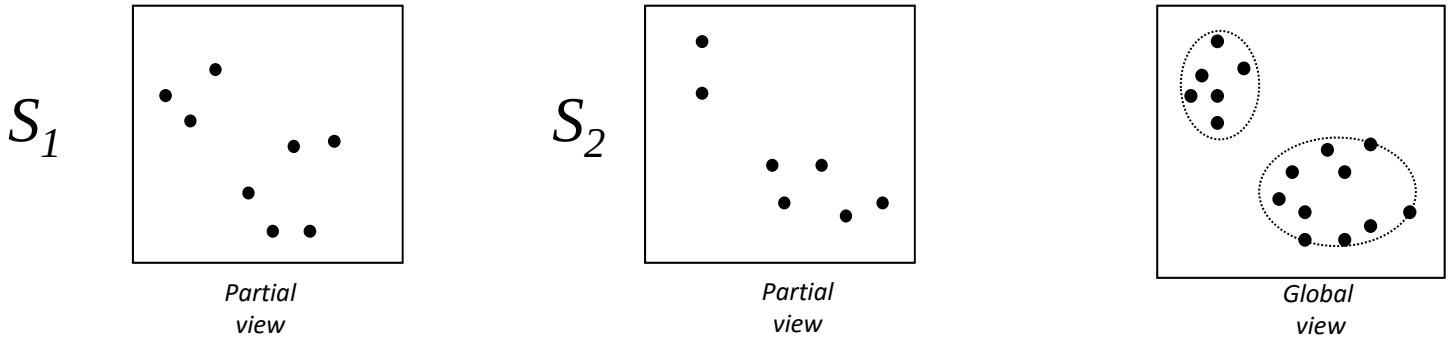
(c) $p \in C_1 \backslash S_1$ is a border point; $p \in C_1$, C_1 and C_2 are NOT merged!

MinPts = 4

Illustration of the relationship between clusters found w.r.t. adjacent space constraints.

Distributed Density-based Clustering

- Arbitrary distribution of points over the sites
- Partitions might spatially overlap
  - Each site S_i might store elements of the ε–range of point p
  - *p* might be a core point, even if p isn‘t a local core point.



- Density-based clustering does not use a compact cluster model

  $\Rightarrow$ Transfer local points to determine global clusters

Distributed Partitioning Clustering

Idea:

- If the cardinality of the transferred points is small, multiple iterations between the sites is not a problem (low traffic)
- *Centroids and cluster quality in k*-Means or related methods is suitable for distributed computing:

$$TD^2 = \sum_{o \in DB} \left(\min_{C_i \in C} \{ d(o, C_i) \} \right)^2 = \sum_{S_j \in DB} \left(\sum_{o \in S_j} \left(\min_{C_i \in C} \{ d(o, C_i) \} \right)^2 \right)$$

Cluster C_i
Partition S_j

- global centroid C_i is computed from local centroids $C_{i,j}$:

$$c_i = \frac{1}{\sum_{C_{i,j} \in DB} |C_{i,j}|} \cdot \sum_{C_{i,j} \in DB} \sum_{o \in C_{i,j}} o$$

- **Summary**: In each iteration it is possible to optimize the global clustering by adding up local components.

Distributed k-Means Pseudocode

Distributed clustering using variance minimization (Master-Slave):

Determine initial distribution and start-centroids

loop:

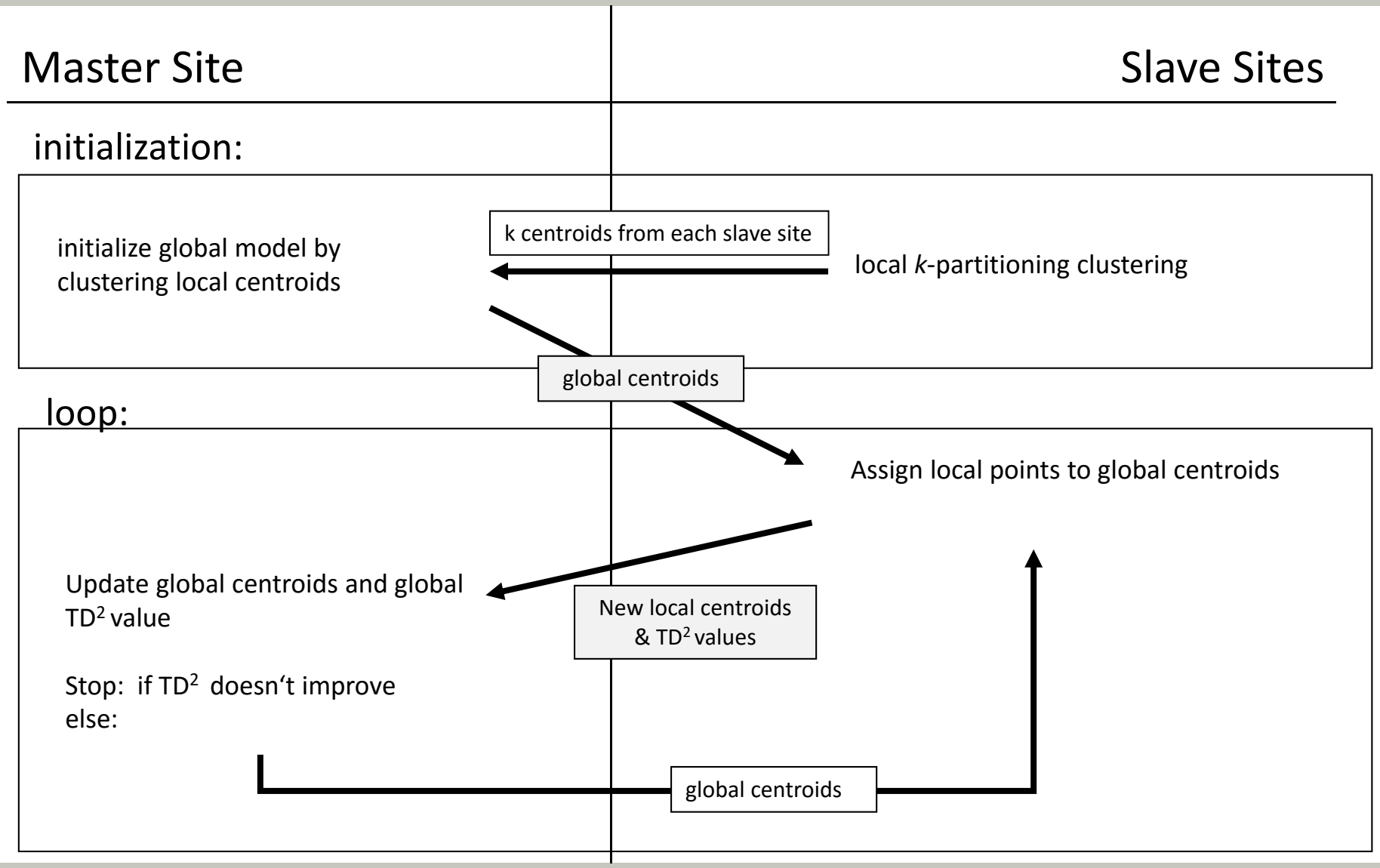
- transfer centroids to all sites
- assign local points to the current centroids
  => compute local centroids and local TD^2 values
- Retransfer local centroids, cluster cardinalities and TD^2 values

$\Rightarrow$ Add local sum-vectors, cluster cardinalities and TD^2 values (implies new global centroids)

$\Rightarrow$ Determine global TD^2 value

if TD^2 value does not improve => terminate

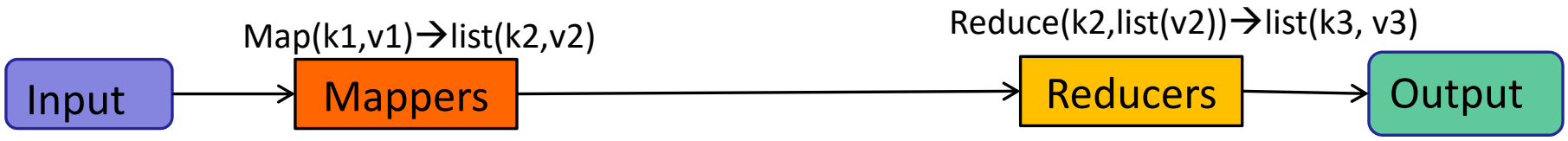
Distributed K-Means Workflow



Parallel Data Analysis with MapReduce

- MapReduce is a programming model for large scale parallel data processing using a large number of computers (nodes), referred to as a cluster.
- Hadoop is an open source implementation of MapReduce.
- Programmers specify the computation in terms of a *map* and a *reduce* function
  - The Map job: takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).
  - The Reduce job: takes the output from a map as input and combines those data tuples into a smaller set of tuples.
  - Both mapper and reducer can be distributed over multiple worker tasks
- Everything else is handled by the execution framework
  - Scheduling: assigns workers to map and reduce tasks
  - "Data distribution": moves processes to data
  - Synchronization: gathers, sorts, and shuffles intermediate data
  - Errors and faults: detects worker failures and restarts
- Optimization is done automatically by adding workers if necessary
- Trade-off: Parallelism vs. Bandwidth

Map and Reduce Tasks



- The Map and Reduce functions of MapReduce are both defined with respect to data structured in (key, value) pairs.
- A Map task perform a transformation, a Reduce task perform an aggregation
- Word count example:
  - Map-Step:

    Example: <ID3,"to be or not to be"> → <to, 1>,<be,1>, <or,1>, <not,1>, <to,1>, <be,1>
  - Shuffle & Sort Step:

    Example: → <to,1>,<to,1><be,1><be,1>,<or,1><not,1>
  - Reduce-Step:

    Example: → <to,2>, <be,2><or,1><not,1>
- For complex problems multiple MapReduce steps might be necessary to implement an algorithm.

Optional Steps

- ***Partitioner***: Controls the distribution of data over the mapper tasks.
  Default: HashPartitioner
- ***Combiner***: Local aggregation step which summarizes data from a mapper
  Step is performed between Map step and shuffle step.
  => Transfer volume from the Mappers to shuffle step can be reduced
  Example: <Today, <1,1,1,1,>> -> <Today,4>

K-Means in MapReduce

Input: A data set *D*, desired number of clusters *k*

Output: *k* centroids minimizing TD^2

$$TD^2(D,C) = \sum_{x \in D} \left(\min_{c \in C} \left(dist(c,x) \right) \right)^2$$

Steps:

- Assign data to cluster centroids
- Compute centroids from a set of objects
- Compute TD^2

$\Rightarrow$ All steps can be done by a linear scan of D

$\Rightarrow$ Results are additive. Cluster centroids and TD^2 are sums and therefore, computable in a distributed way. (associative law)

k-Means using MapReduce


```
Master:
   Sample k initial centroids C.
   WHILE TD² < oldValue
       oldValue = TD²
       assign points in D to centroids in C (Mapper)
       compute centroids C and quality TD2 (Reducer )
   RETURN C
```

Remark:

- Only the expensive steps are processed in a distributed way
- One MapReduce task for each iteration
- C has to be transferred to mappers and reducers

K-Means: Map Step


```
Input: D: dataset,C: set of centroids, k=|C|: # centroids
Output: <centroid_id, instance>

FOR EACH instance  v in D DO
   bestCluster = null; minDist = ∞
   FOR EACH C_i in C DO
       IF minDist > dist(C_i, v) THEN
              minDist = dist(C_i, v)
              bestCluster = C_i
       ENDIF
   END FOR
   OUTPUT<bestCluster, v>
END FOR
Shuffle and sort: →  <Cluster,<v_1,..,v_l>>
```

Mapper: Assign points to centroids

K-Means: Reduce Step


```
Input: <Cluster,<v1,..,vl>> from previous step
Output: <newC, TD2> new centroids and partial TD2

FOR EACH <C, <v1,..,vl>>  DO
   linearSum = 0;
   count = 0;
   TD2 = 0;
   FOR EACH v in <v1,..,vl> DO
       linearSum += v
       count = count+1
       TD2 +=dist(v,C)
   END FOR
   newC = linearSum/count
   OUTPUT< newC, TD2>
END FOR
```

Reducer: compute new cluster centers and their quality

K-Means using MapReduce

- Number of calls corresponds to the number of iterations
- Optimization by local combiners which precompute parts of the linear sums.
- Algorithms does not solve the problem of a suitable initialization
- Newer methods use sampling techniques to cluster data in sublinear time.

Summary Parallel and Distributed Data Mining

- Modern hardware developments can speed up Data Mining
- Large scale data mining is necessary nowadays due to the amount and complexity of collected data.
- Solutions should be: scalable, incremental and interactive

- The raise of the parallel computing recently, is reshaping the area
  - MapReduce, Apache Mahout, Apache SPARK (Mlib), Apache STORM,…

References

- [XuJaeKri99] X. Xu, J. Jäger, and H.-P. Kriegel, *A Fast Parallel Clustering Algorithm for Large Spatial Databases*, DMKD 3(3), pp. 263-290, 1999.
- W. Zhao, H. Ma, and Q.He, *Parallel K-Means Clustering Based on MapReduce*, CloudCom 2009.
- H. Xiao, *Towards Parallel and Distributed Computing in Large-Scale Data Mining: A Survey*, TR TUM, 2010.
- M. Zaki, *Parallel and Distributed Data Mining: An Introduction*, Large-Scale Parallel Data Mining, Springer, 2002.
- Zhao W., Ma H., He Q.:Parallel K-Means Clustering Based on MapReduce, CloudCom, (pp 674-679) 2009
- Lammel R.: *Google‘s MapReduce Programming Model- Revisited*. Science fo Computer Programming 70, 1-30 , 2008
- Ene A., Im S., Moseley B.: *Fast Clustering using MapReduce*, 17th int. Conf. on Knowledge Discovery and Data Mining (KDD‘11), 2011