Distributed Processing of Data Streams and Large Data Sets

DEIS'10
Advanced School on Data Exchange, Integration, and Streams
Nov 11, 2010, Schloss Dagstuhl

DATA MANAGEMENT AND DATA EXPLORATION GROUP
Prof. Dr. rer. nat. Thomas Seidl
RWTH AACHEN UNIVERSITY

Marwan Hassani
Data Management and Data Exploration Group
RWTH Aachen University, Germany

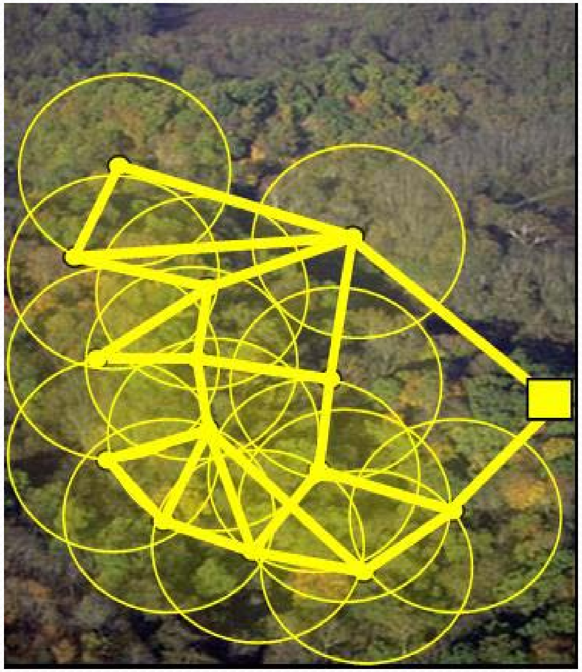
Agenda

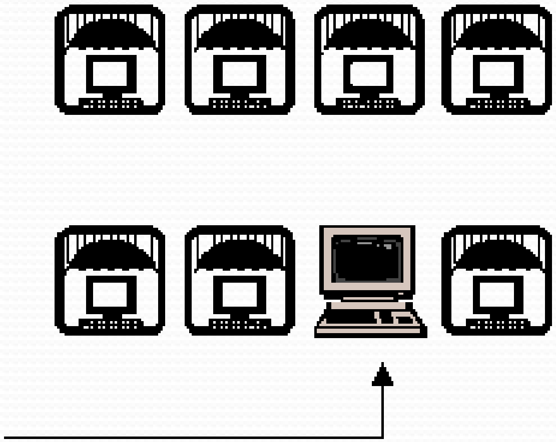
- Distributed Stream Processing (DSP) Systems
- Examples on Continuous DSP Systems
- Distributing Computations of Large Data Sets
  - Mud algorithms as a model for MapReduce-like frameworks

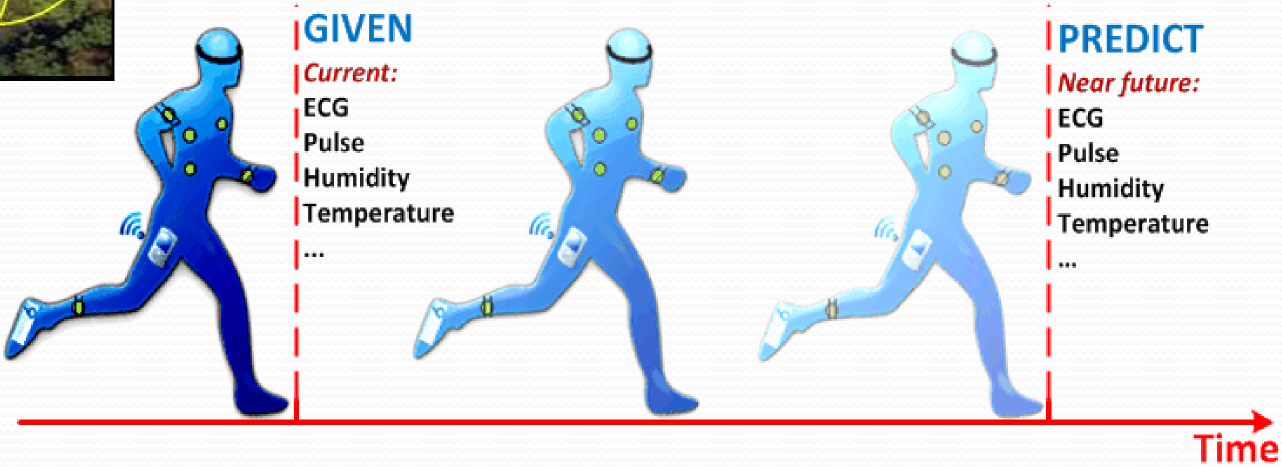
Agenda

- Distributed Stream Processing (DSP) Systems
- Examples on Continuous DSP Systems
- Distributing Computations of Large Data Sets
  - Mud algorithms as a model for MapReduce-like frameworks

Distributed Stream Processing Systems







Distributed Stream Processing Systems

- Stream processing systems: manage multiple parallel stream data originated from physically distributed sources (e.g. IP monitoring)

- Centralized stream processing systems use algorithms that ignore communication-efficiency issues

Categorization of Distributed Stream Processing System

- **W.r.t. the class of queries applied over systems**
  - (Non)-Holistic aggregates
  - Duplicate-(in)sensitive aggregates
- **W.r.t. the communication model**
  - Hierarchical
  - Fully-distributed
- **W.r.t. the querying model**
  - One-Shot DSP Systems
  - Continuous DSP Systems

Non-holistic vs. Holistic Aggregates

- In non-holistic aggregates (e.g. `MIN`, `MAX`, `AVERAGE`): partial answers over a subset of streams are usable for final answers

- In holostic aggregates (e.g. `MEDIAN`): no useful partial aggregates can be done, all the data must be brought together to perform the aggregate. The introduce more challenges for designing the DSP

Categorization of Distributed Stream Processing System

- **W.r.t. the class of queries applied over systems**
  - (Non)-Holistic aggregates
  - Duplicate-(in)sensitive aggregates
- **W.r.t. the communication model**
  - Hierarchical
  - Fully-distributed
- **W.r.t. the querying model**
  - One-Shot DSP Systems
  - Continuous DSP Systems

Duplicate: Sensitive vs. Insensitive Aggregates

- Duplicate-insensitive aggregates (e.g. `MIN, Count Distinct`): are unaffected by duplicate readings from a single site

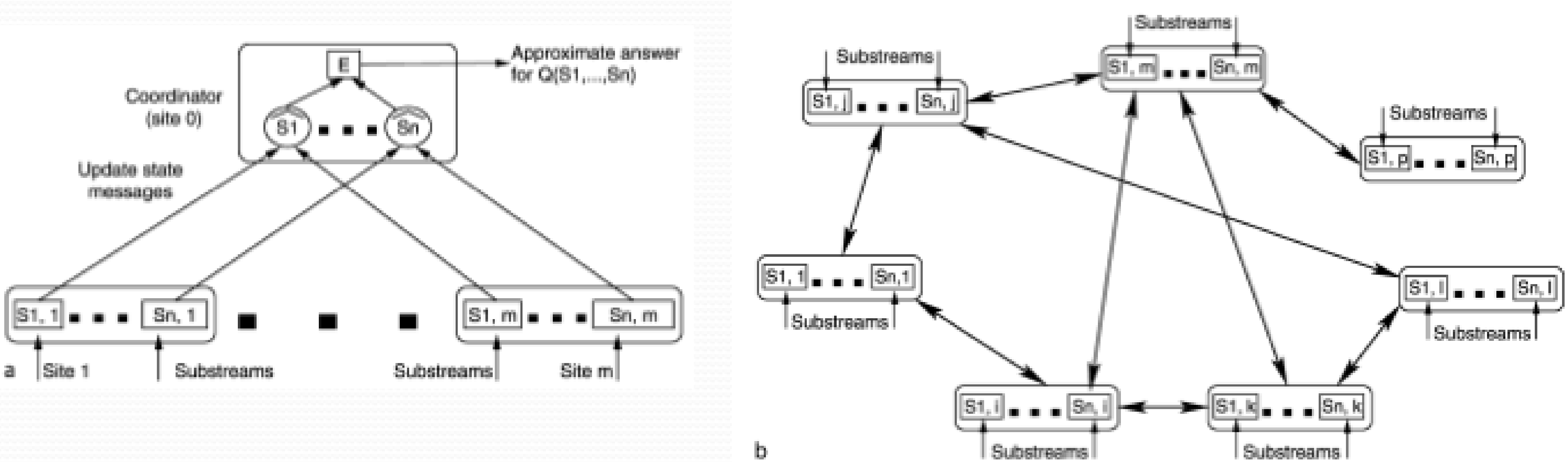
- Duplicate-sensitive aggregates (e.g. `SUM`, `top-k`): will change when a duplicate reading is reported. They demand more robust DSP system

Categorization of Distributed Stream Processing System

- **W.r.t. the class of queries applied over systems**
  - **(Non)-Holistic aggregates**
  - **Duplicate-(in)sensitive aggregates**
- **W.r.t. the communication model**
  - **Hierarchical**
  - **Fully-distributed**
- **W.r.t. the querying model**
  - **One-Shot DSP Systems**
  - **Continuous DSP Systems**

Hierarchical vs. Fully Distributed DSP Systems

- The characteristics of underlying network communication protocol have an impact on the design of the DSP system



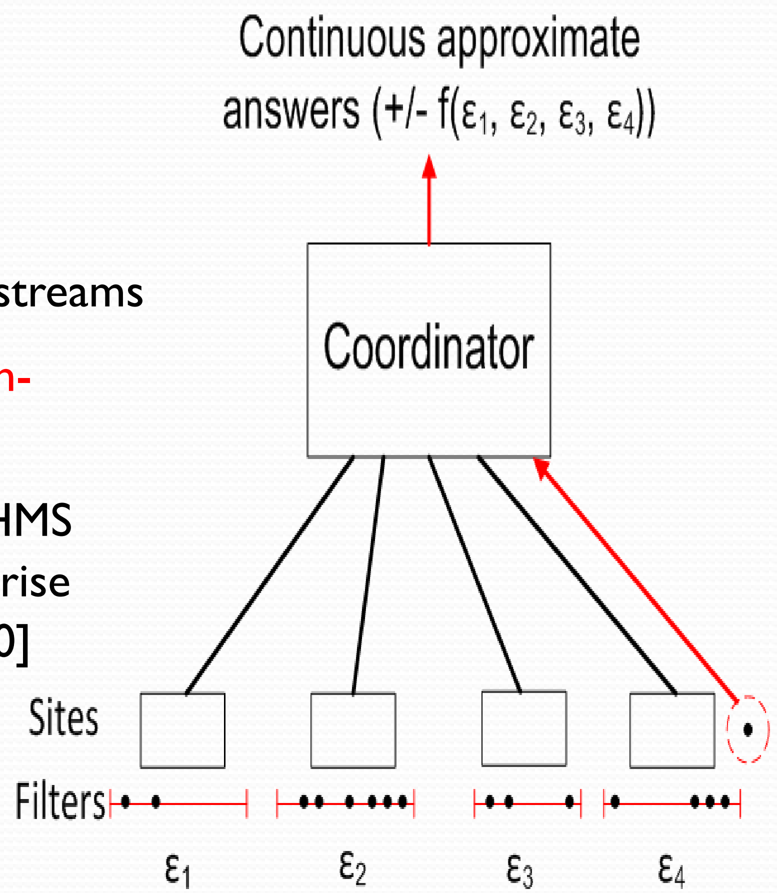
- One coordinator is responsible for answering queries, robustness is key concern
- No centralized authority, the goal is having an agreement on the answer of a query

Taxonomy of Distributed Stream Processing System

- **W.r.t. the class of queries applied over systems**
  - (Non)-Holistic aggregates
  - Duplicate-(in)sensitive aggregates
  - Complex correlation queries
- **W.r.t. the communication model**
  - Hierarchical
  - Fully-distributed
- **W.r.t. the querying model**
  - One-Shot DSP Systems
  - Continuous DSP Systems

One-Shot vs. Continuous DSP Systems

- Continuous DSP Systems:
  - ➔Remote sites must collaborate to continuously maintain a query answer that describes (within specified error bound) the current state of the streams
  - ➔Approximation is used to design communication-efficient solutions
  - ➔Applications: Monitoring in sensor networks [HMS SensorKDD '09, HM+, SensorKDD '10] enterprise network security (intrusion detection) [HS 2010]



- One-Shot DSP Systems:
  - ➔Initiated by user queries
  - ➔TAG is a tree-based aggregation system for sensor networks given by [Madden et. al., OSDI 2002]

Agenda

- Distributed Stream Processing (DSP) Systems
- Examples on Continuous DSP Systems
- Distributing Computations of Large Data Sets
  - Mud algorithms as a model for MapReduce-like frameworks

Motivating Scenario

Application [Akyiyildiz et al. 2005]

- Let m sensor nodes be **distributed** in an underwater acoustic monitoring system
- **Task**: each node keeps track of certain school of fishes based on a given wave length and reports the results to a central base stations
- The base station maintains a k-clustering of the schools
- **Target**: deploying k attracting or dispelling acoustic devices near the k center points to use minimum energy for covering the whole region

Motivating Scenario

Application [Akyiyildiz et al. 2005]

- Let m sensor nodes be **distributed** in an underwater acoustic monitoring system
- **Task**: each node keeps track of certain school of fishes based on a given wave length and reports the results to a central base stations
- The base station maintains a k-clustering of the schools
- **Target**: deploying k attracting or dispelling acoustic devices near the k center points to use minimum energy for covering the whole region

Settings

- Underwater sensor networks are a particularly resource constrained because of physical conditions (reduced channel capacity, harsh environment).

Motivating Scenario

Application [Akyiyildiz et al. 2005]

- Let m sensor nodes be **distributed** in an underwater acoustic monitoring system
- **Task**: each node keeps track of certain school of fishes based on a given wave length and reports the results to a central base stations
- The base station maintains a k-clustering of the schools
- **Target**: deploying k attracting or dispelling acoustic devices near the k center points to use minimum energy for covering the whole region

Settings

- Underwater sensor networks are a particularly resource constrained because of physical conditions (reduced channel capacity, harsh environment).
- Nodes: **unattached** for a long time [or not at all] (lifetime= battery lifetime)

K-Center Clustering

Offline approach

- Given a group $|P| = n$, find: $k \leq n$ **centers** for disks with **smallest radius** R to cover all $p \in P$
- Out of $\binom{n}{k}$ possible ways, find the one which minimizes the cost
- NP-hard to find better than 2-**approximation** to the optimal clustering [Feder et al., 1988]
- Approximation solutions to the optimal clustering are seeked

K-Center Clustering

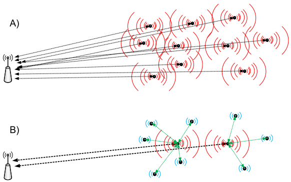
Offline approach

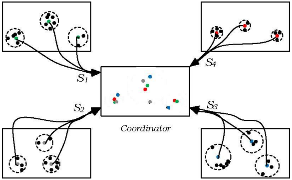
- Given a group $|P| = n$, find: $k \leq n$ **centers** for disks with **smallest radius** R to cover all $p \in P$
- Out of $\binom{n}{k}$ possible ways, find the one which minimizes the cost
- NP-hard to find better than 2-**approximation** to the optimal clustering [Feder et al., 1988]
- Approximation solutions to the optimal clustering are seeked

Main idea of the online (incremental) approach

- For the **current** points in the sliding window of the **stream** points, find a **current** solution $S = \{c_1, c_2, \ldots, c_k, R\}$
- **Continuously** updates S to keep it valid as the stream evolves

Distributed *k*-center Clustering





Suggested Global Clustering

- The coordinator receives *k*-center clusterings from *m* sites and forwards that to a far base station
- Estimate the residual energy of nodes $\Rightarrow$ change coordinator

Global Parallel Guessing [Cormode et al., ICDE 2007]

The PG Algorithm (Initialization phase, performed on the coordinator)

- Pick an arbitrary point as the first center $C = \{c_1\}$
- Get a *big enough* initialization sample I from the stream
- Since we do not know R in advance, we make multiple guessing of R as $(1 + \frac{\varepsilon}{2}), (1 + \frac{\varepsilon}{2})^2, (1 + \frac{\varepsilon}{2})^3, \cdots$ for $0 < \varepsilon < 1$
- Drop guesses that are smaller than $min_{p,q \in P} d(p, q)$
- Also drop guesses that are larger than $max_{p,q \in P} d(p, q)$
- Run the algorithm in parallel on each of these radii like this:

  while $|C| < k$

  For each $p \in I$ compute: $r_p = min_{c \in C} d(p, c)$

  If $r_p > R \Rightarrow C = C \cup \{p\}$
- Drop guesses that result in more than k centers
- Store the resulted $\{c_{1i}, c_{2i}, \ldots, c_{ki}, R_i\}$ for each guess R_i

Global Parallel Guessing [Cormode et al., ICDE 2007]

The PG Algorithm (Running Phase, on the site side)

1. While there is input stream point p compute: $r_p = min_{c \in C} d(p, c)$
2.   If $r_p > R$ Then
3.   If $|C| < k$ Then
4. $C = C \cup \{p\}$
5.   update the coordinator with the new center
6.   else
7.   ask the coordinator for a new (bigger) guess of R
9. end while

The PG Algorithm (Running Phase, on the coordinator side)

1. Consider one global guess R_{global} picked from the guesses for all sites
1. Whenever there is a request for a bigger R from site m
2.   update m with a R_{global}

Clustering Quality and Complexity

Clustering quality and storage demand

- $(2+\varepsilon)-$approximation to optimal clustering is guaranteed
- Stores at most $O(\frac{k}{\varepsilon} log\Delta)$ $(\Delta = max_{p,q\in P} d(p,q)/min_{p,q\in P, p\neq q} d(p,q))$
- Recent work from [Guha, EDBT 2009] presented a **centralized**, $2(1+\varepsilon)-$approximation version using $O(\frac{k}{\varepsilon} log\frac{1}{\varepsilon})$ space

Clustering Quality and Complexity

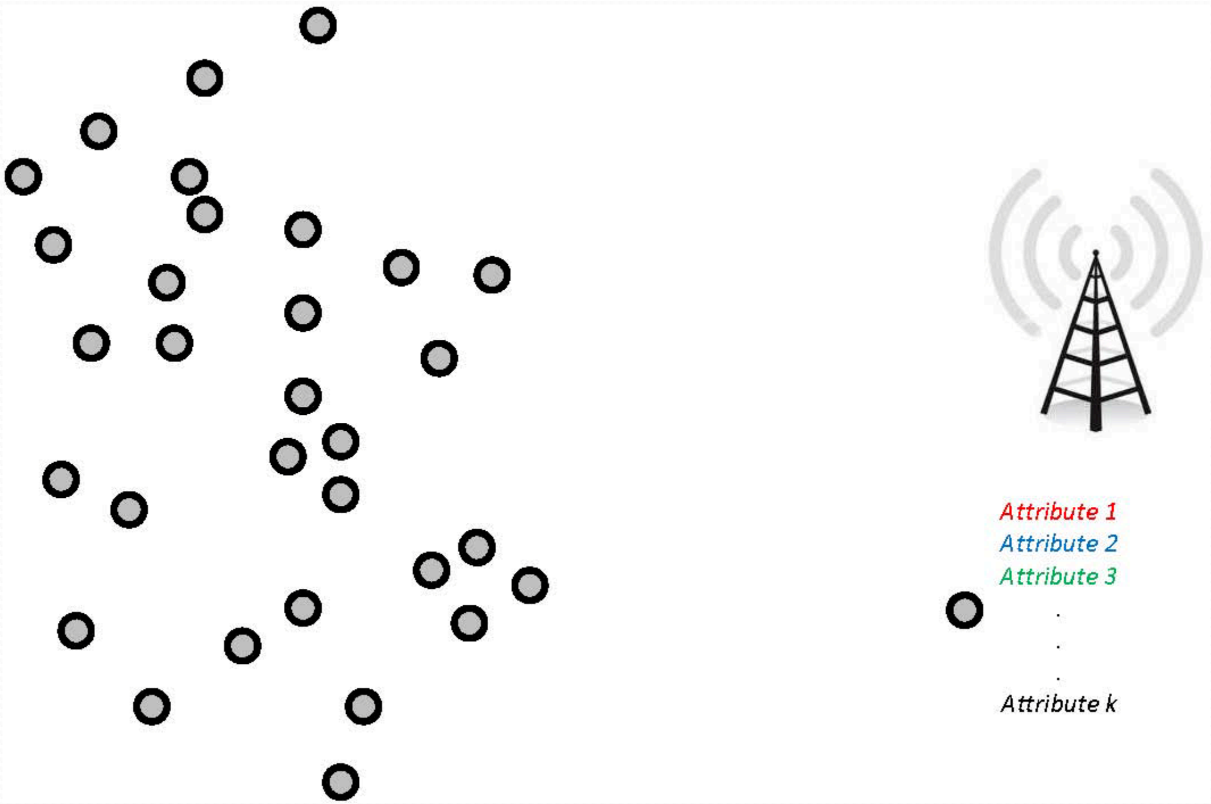
Clustering quality and storage demand

- $(2+\varepsilon)-$approximation to optimal clustering is guaranteed
- Stores at most $O(\frac{k}{\varepsilon} log\Delta)$ ($\Delta = max_{p,q \in P} d(p,q) / min_{p,q \in P, p \neq q} d(p,q)$)
- Recent work from [Guha, EDBT 2009] presented a **centralized**, $2(1+\varepsilon)-$approximation version using $O(\frac{k}{\varepsilon} log \frac{1}{\varepsilon})$ space

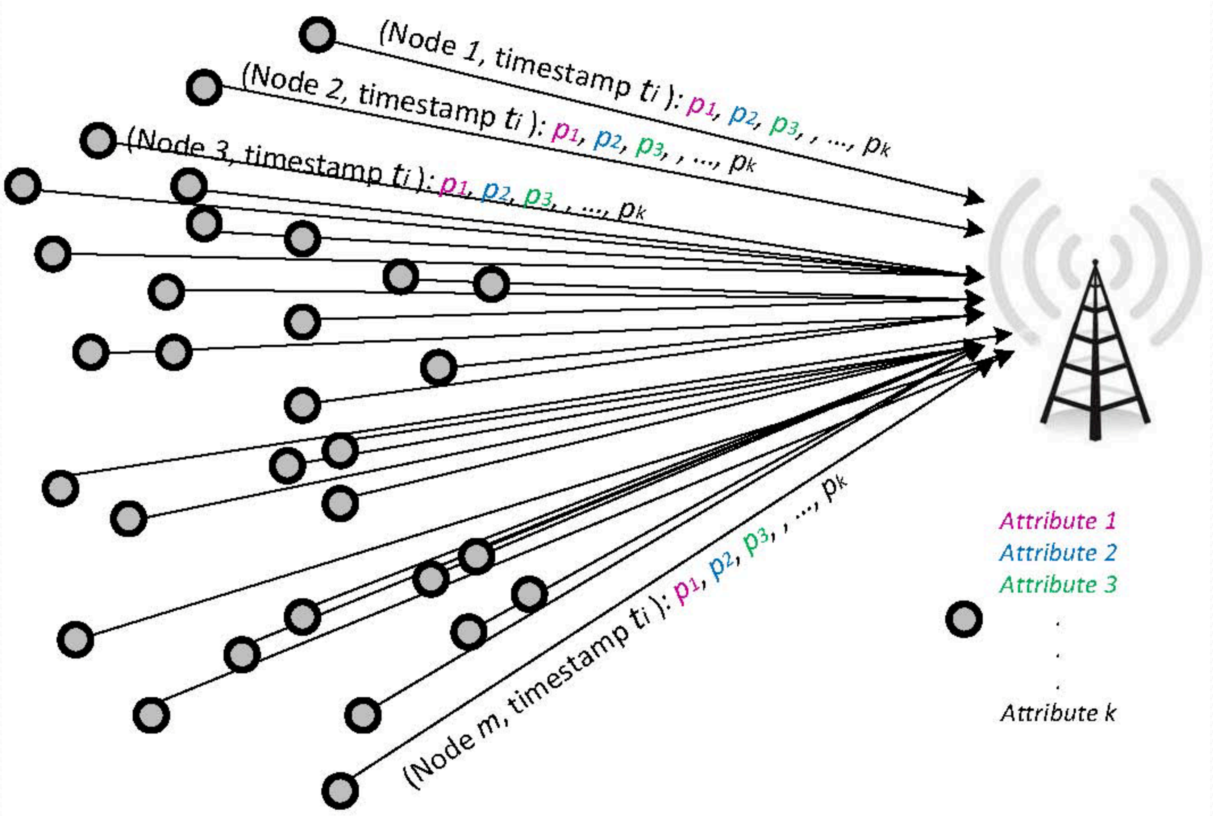
The Communication Complexity

- Worst case: all *m* nodes simultaneously observe a new non-covered point *p* for a guess *R* and send an update request to the coordinator
- This results in updating *k* centers for each guess, there are at most $O(\frac{1}{\varepsilon} log\Delta)$ guesses
- The communication cost is $O(\frac{km}{\varepsilon} log\Delta)$

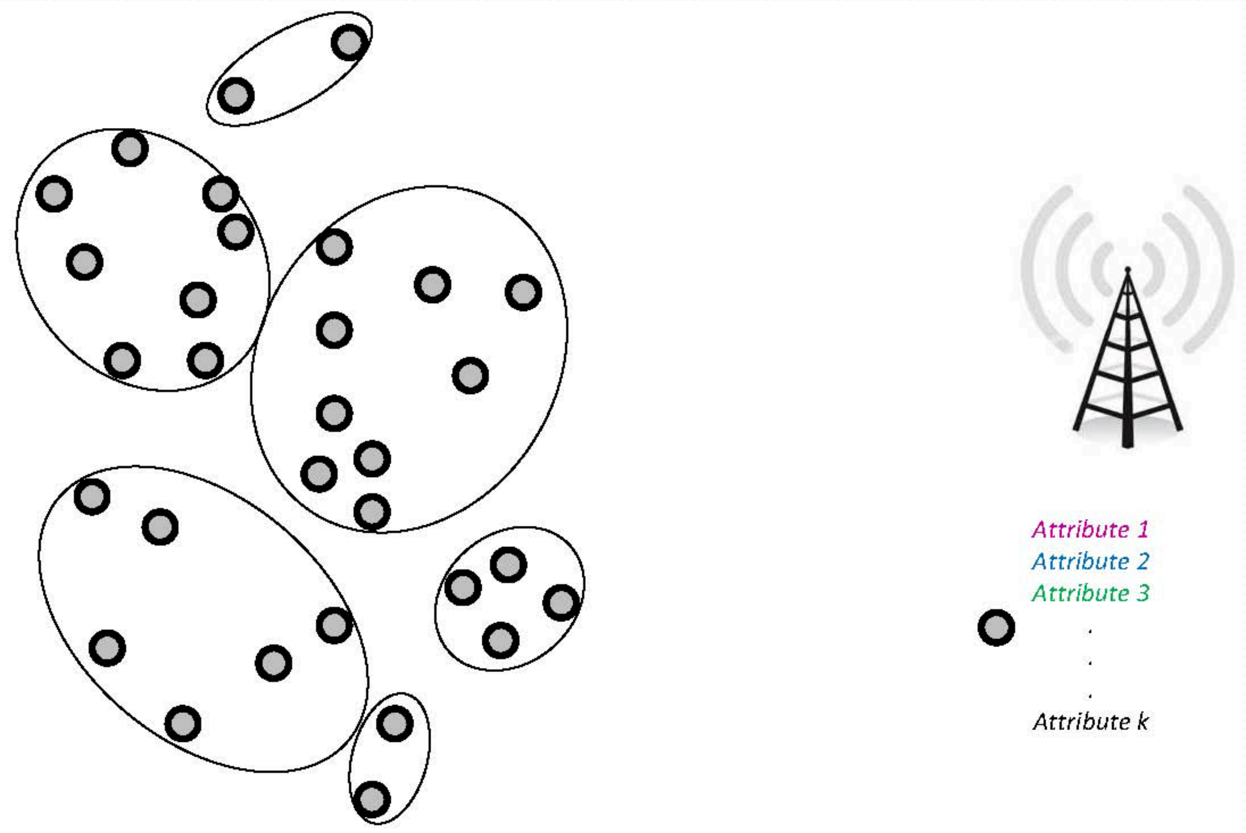
Collecting Sensor Data



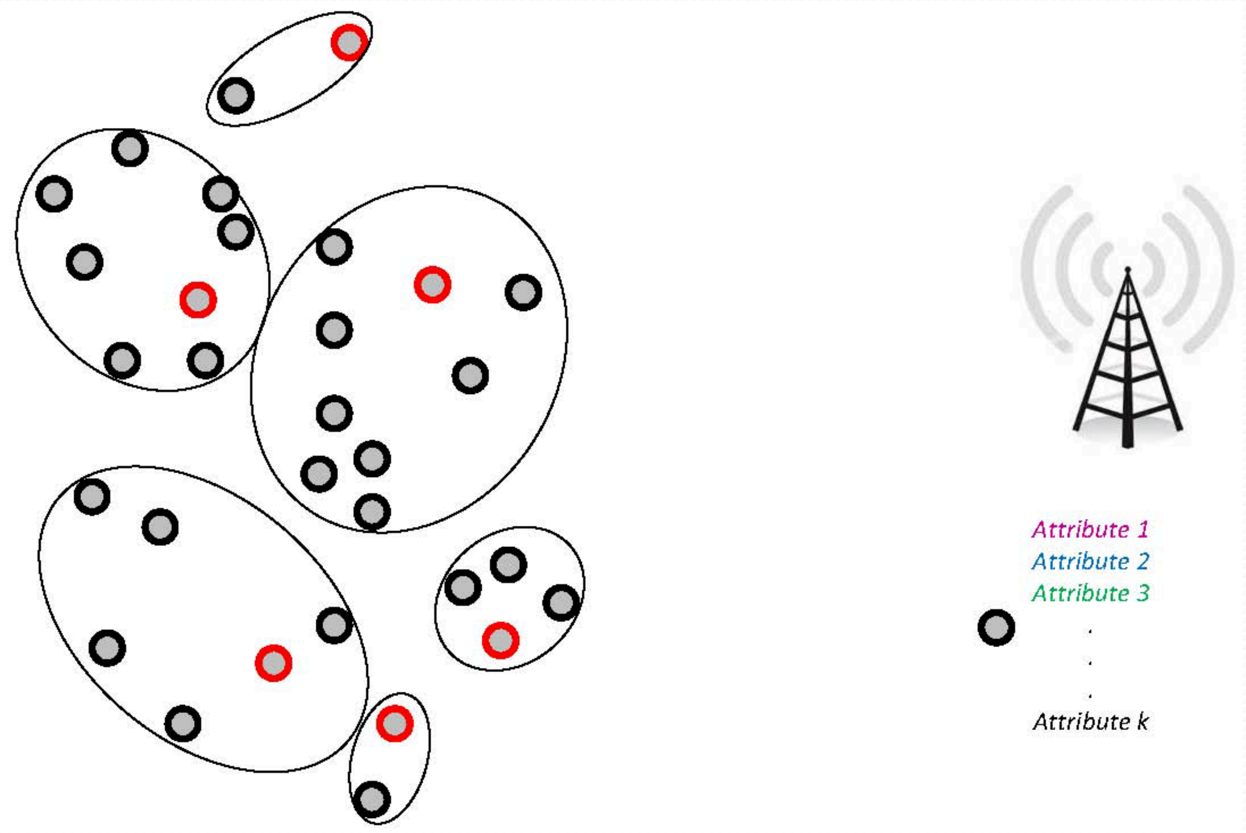
Collecting Sensor Data



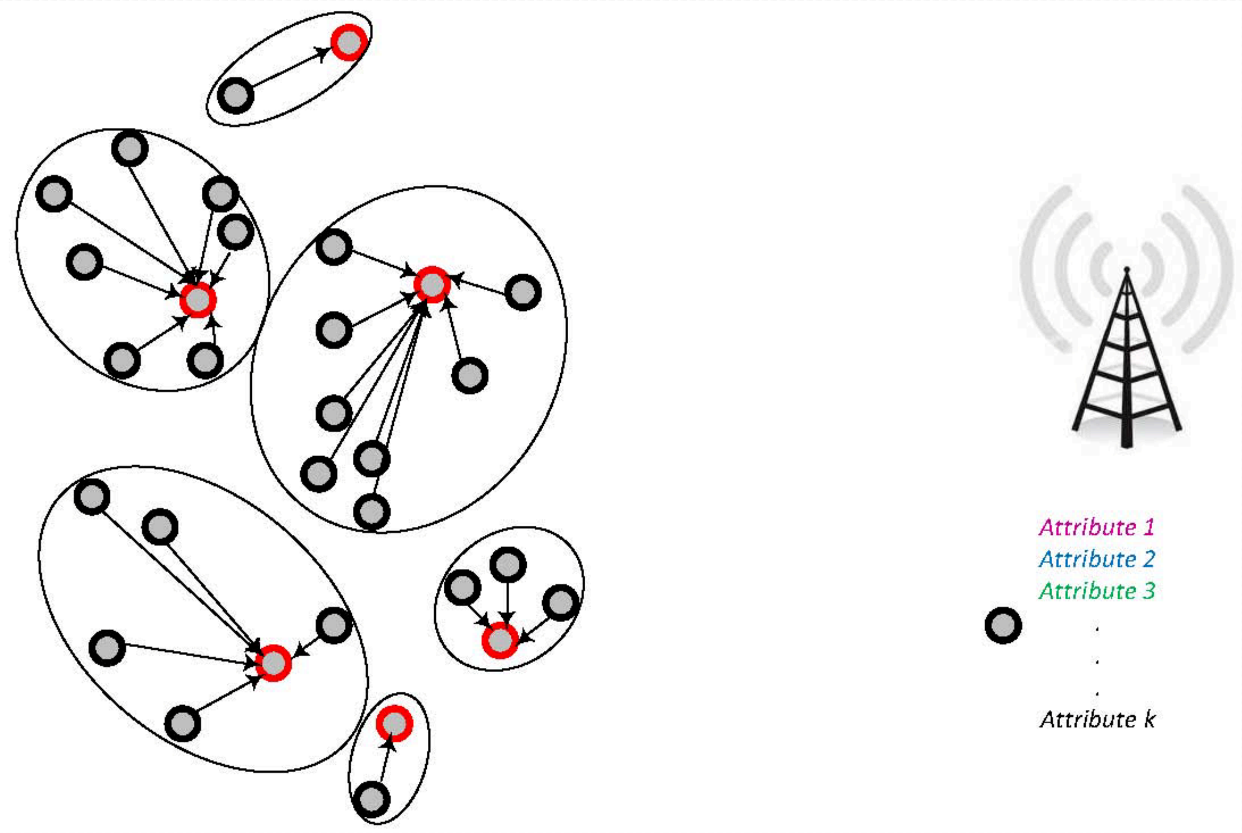
Better: group the neighbours



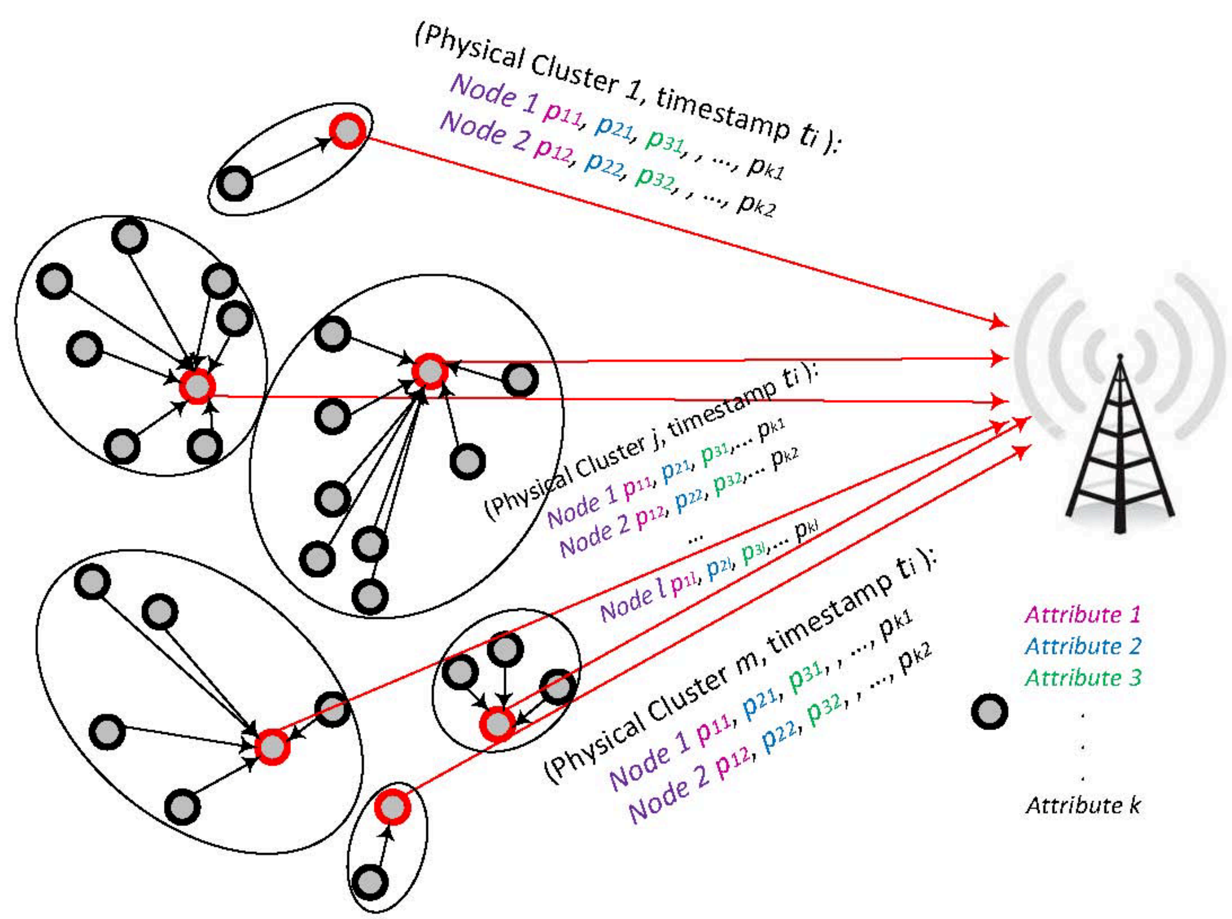
Select coordinators



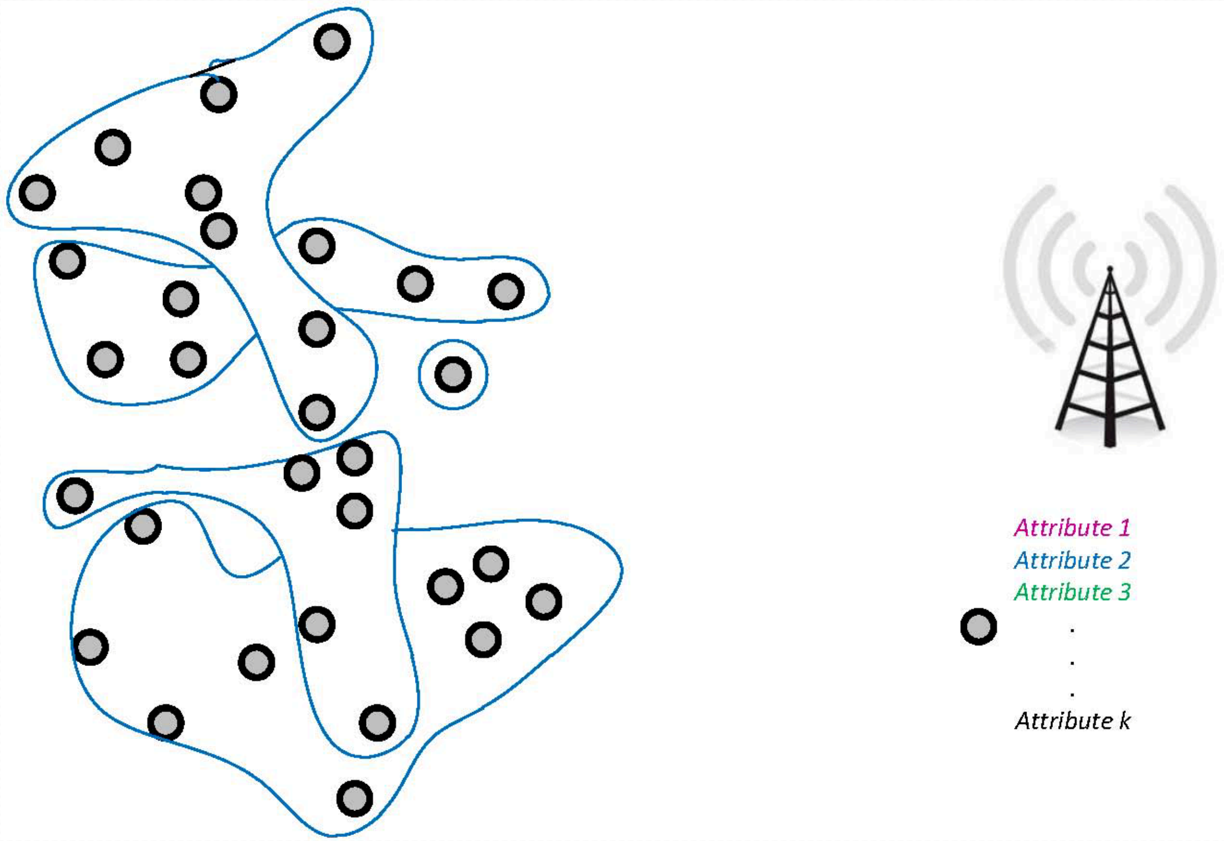
Let cluster members send their readings locally to coordinators



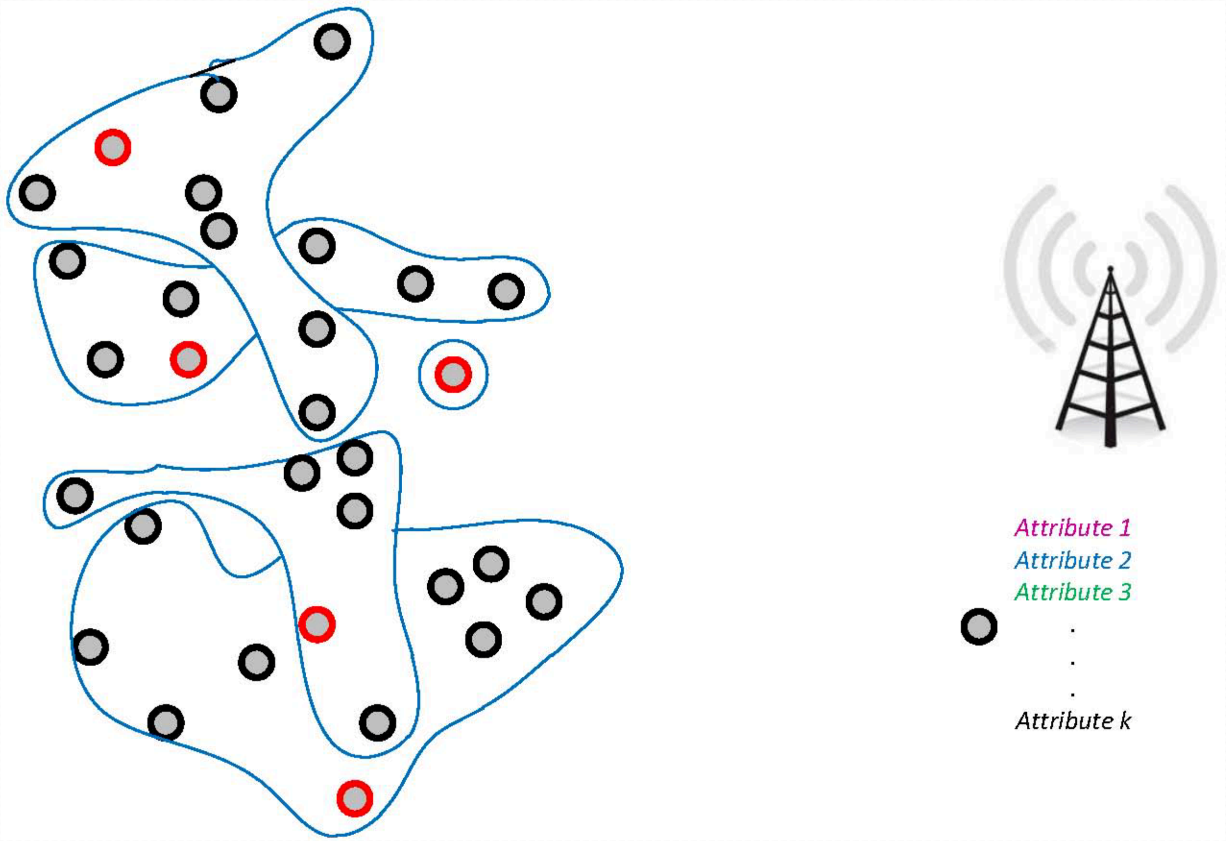
And let coordinators forward it to the base station



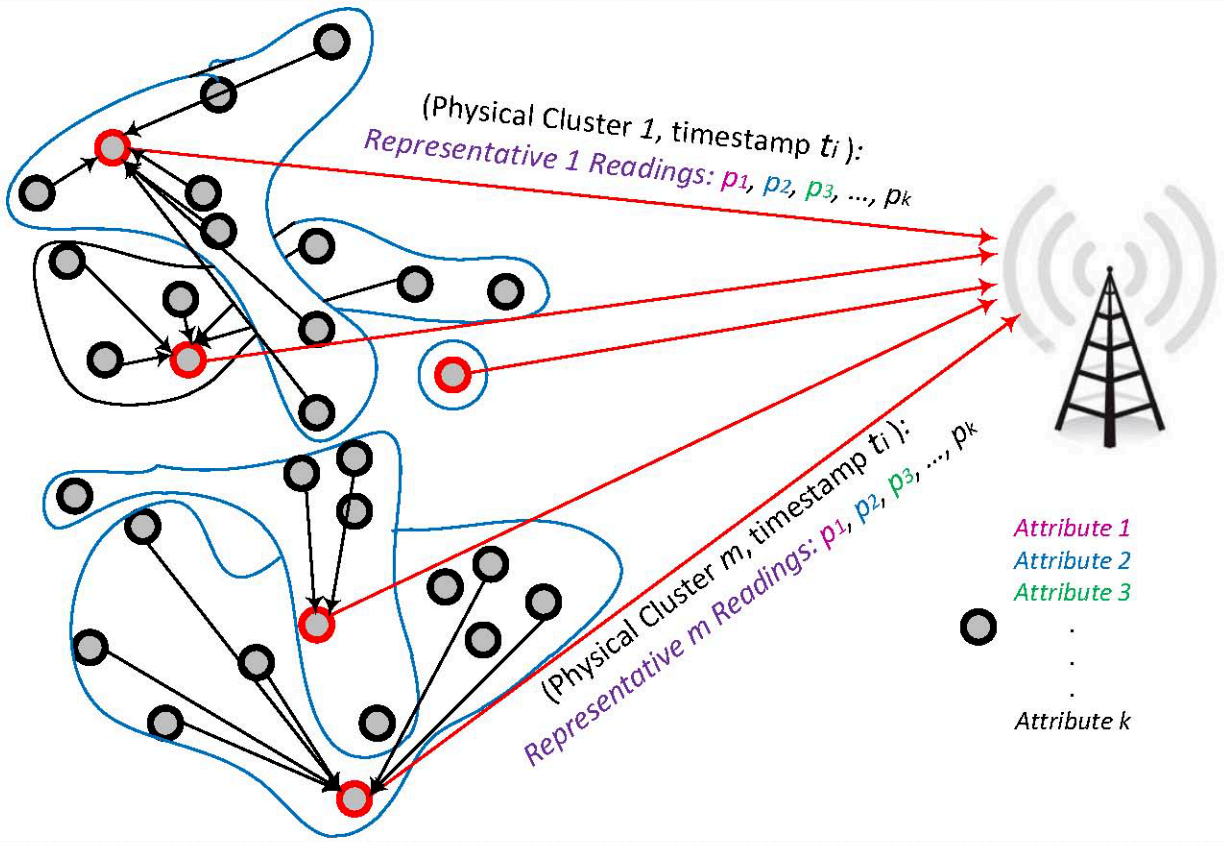
Even better: let the grouping depend on the similarity of sensed data



Then select the best representative of each physical cluster

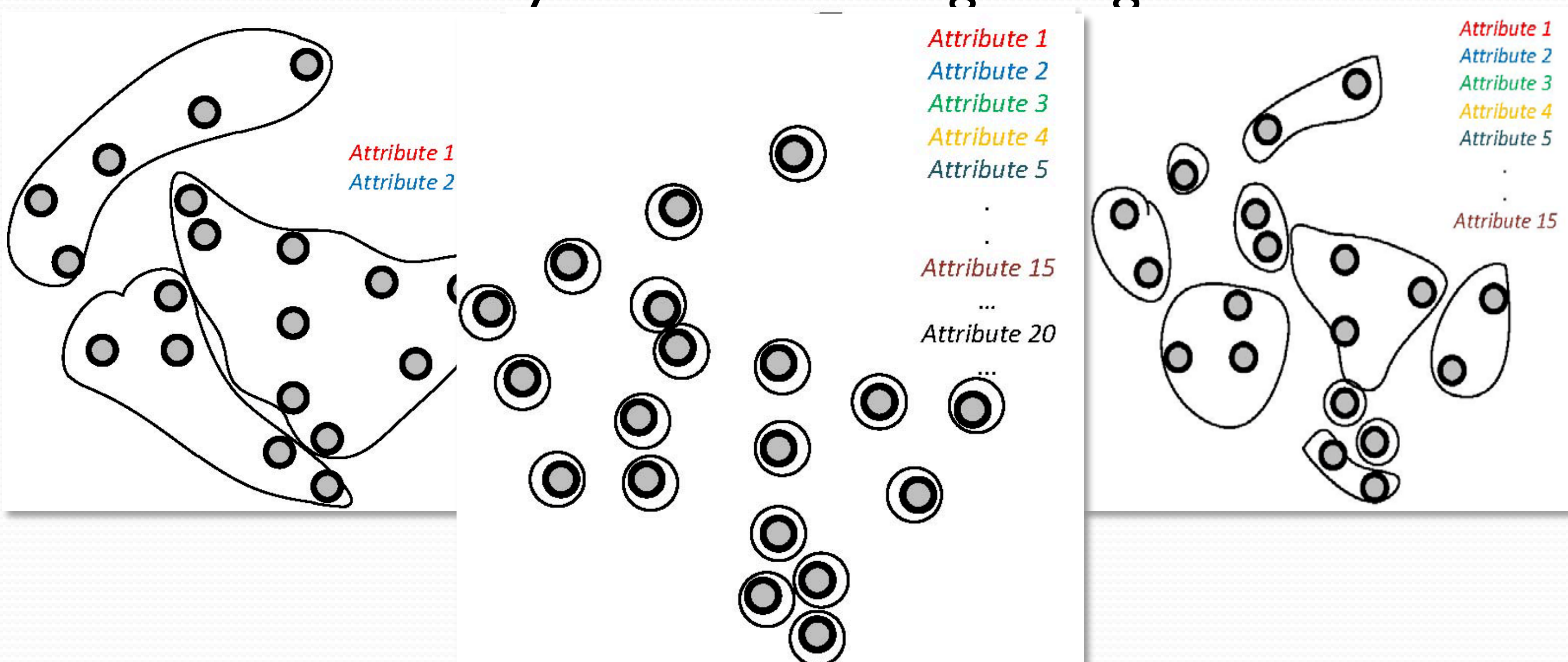


Use only the readings of the representatives to update the base station of the status of the whole network

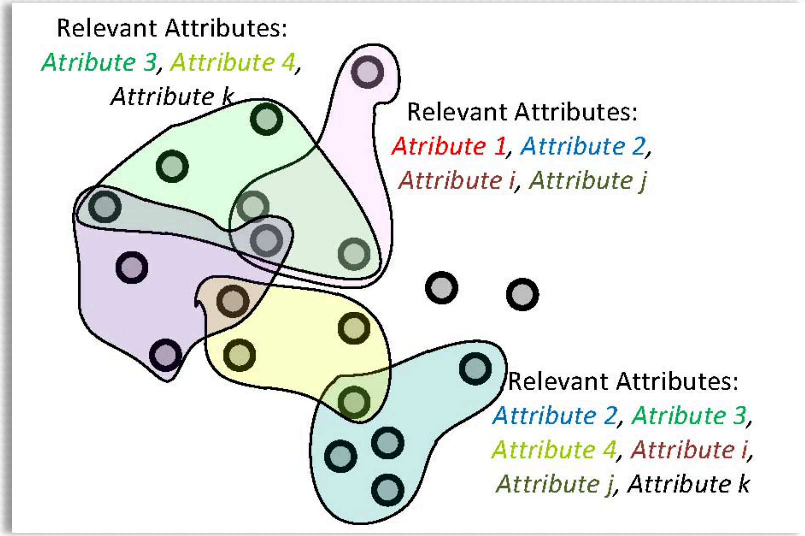


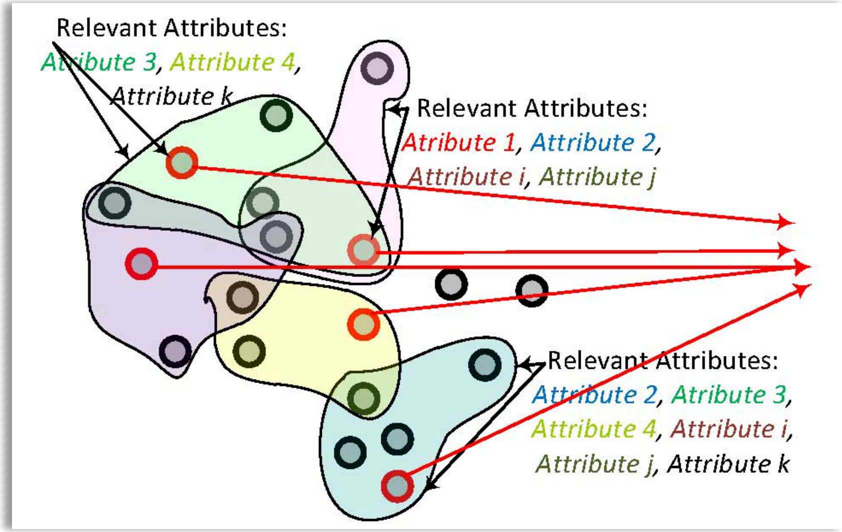
Curse of Dimensionality

- The number of nodes sensing similar data decreases as the dimensionality of sensed data gets higher



The ECLUN* Algorithm





* - Hassani et al. . In SsensorKDD'10

Agenda

- Distributed Stream Processing (DSP) Systems
- Examples on Continuous DSP Systems
- Distributing Computations of Large Data Sets
  - Mud algorithms as a model for MapReduce-like frameworks

Motivation

- Processing large data sets
- Single-pass streaming systems are ideal for rapidly processing items in such data sets using local storage
- With truly massive data like logs of internet activity, stream algorithms are not sufficient.

- The input size in such applications is so big that no single processor can perform even a single pass over it in a reasonable time
- The solution is to distribute the computation over different sites

Challenges when Distributing Computations of Large Data Sets

- Designing a distributed version of data processing algorithms
- Communication cost amongst sites (communication efficiency)
- Load balancing between sites
- Availability in the presence of failure

MapReduce

- a programming model and an associated implementation for processing and generating large datasets
- Applicable to a variety of real-world tasks
- Users specify the computation using *map* and *reduce* functions
- The underlying runtime system automatically:
  1. Parallelizes the computation across large-scale clusters and machines
  2. Handles machine failures
  3. Schedules inter-machines communication for efficient use of network and disks
- Easy, widely used. On Google clusters daily:
  - 10^5 jobs executed
  - 20+ petabytes of data processed

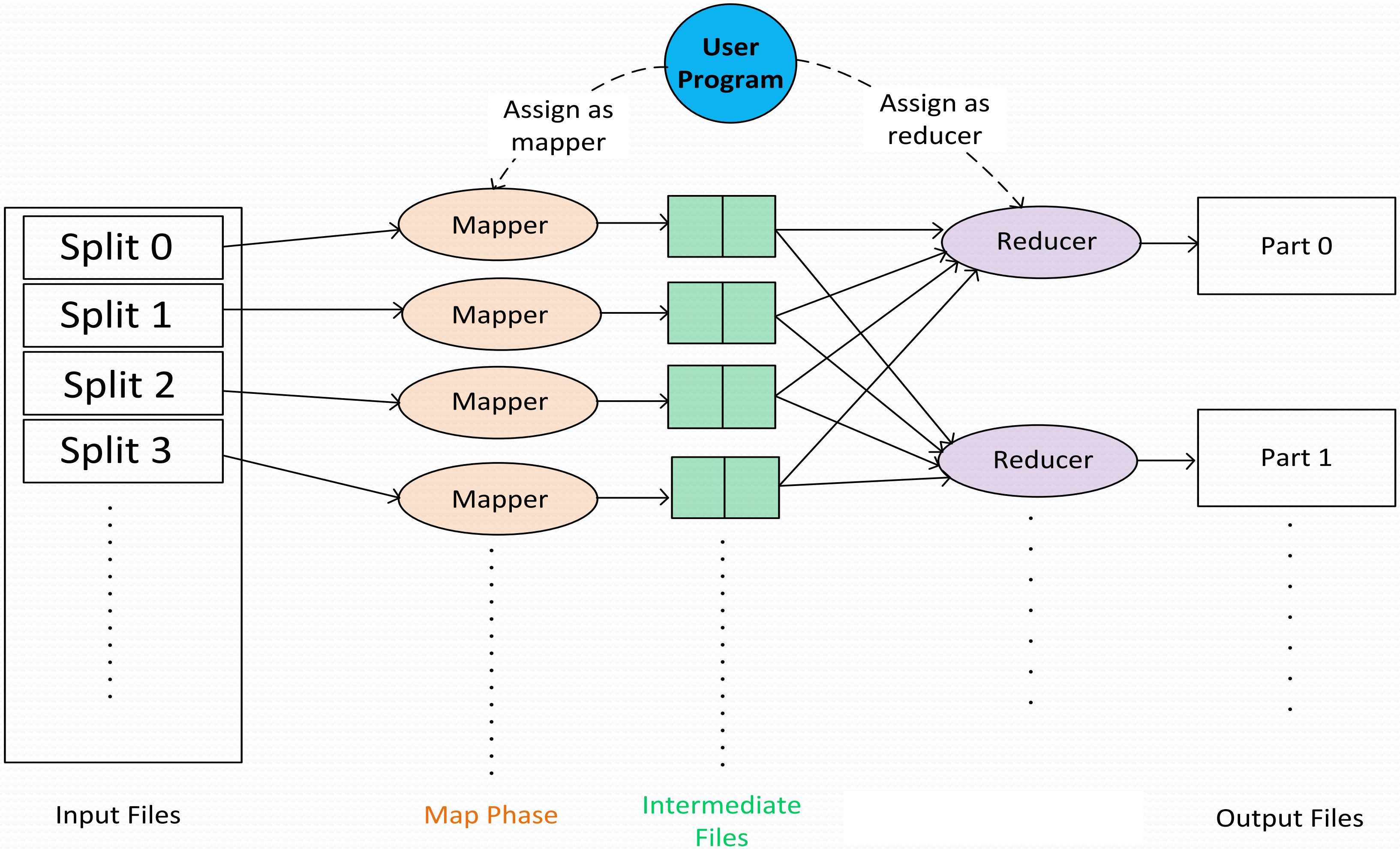
MapReduce

employees.txt

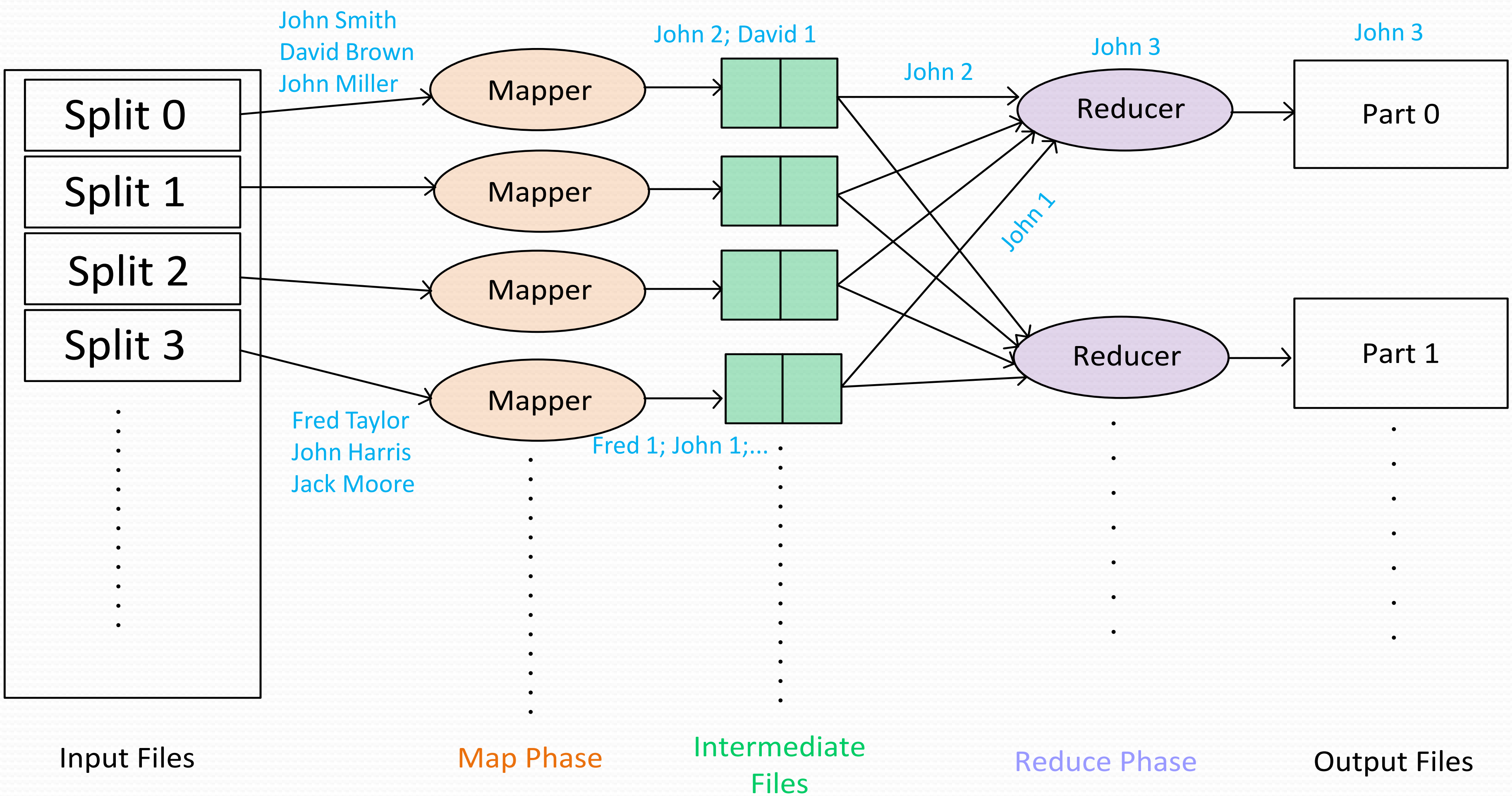
# LAST	FIRST	SALARY
Smith	John	$90,000
Brown	David	$70,000
Johnson	George	$95,000
Yates	John	$80,000
Miller	Bill	$65,000
Moore	Jack	$85,000
Taylor	Fred	$75,000
Smith	David	$80,000
Harris	John	$90,000
...	...	...
...	...	...

Q: "What is the frequency of each first name?"

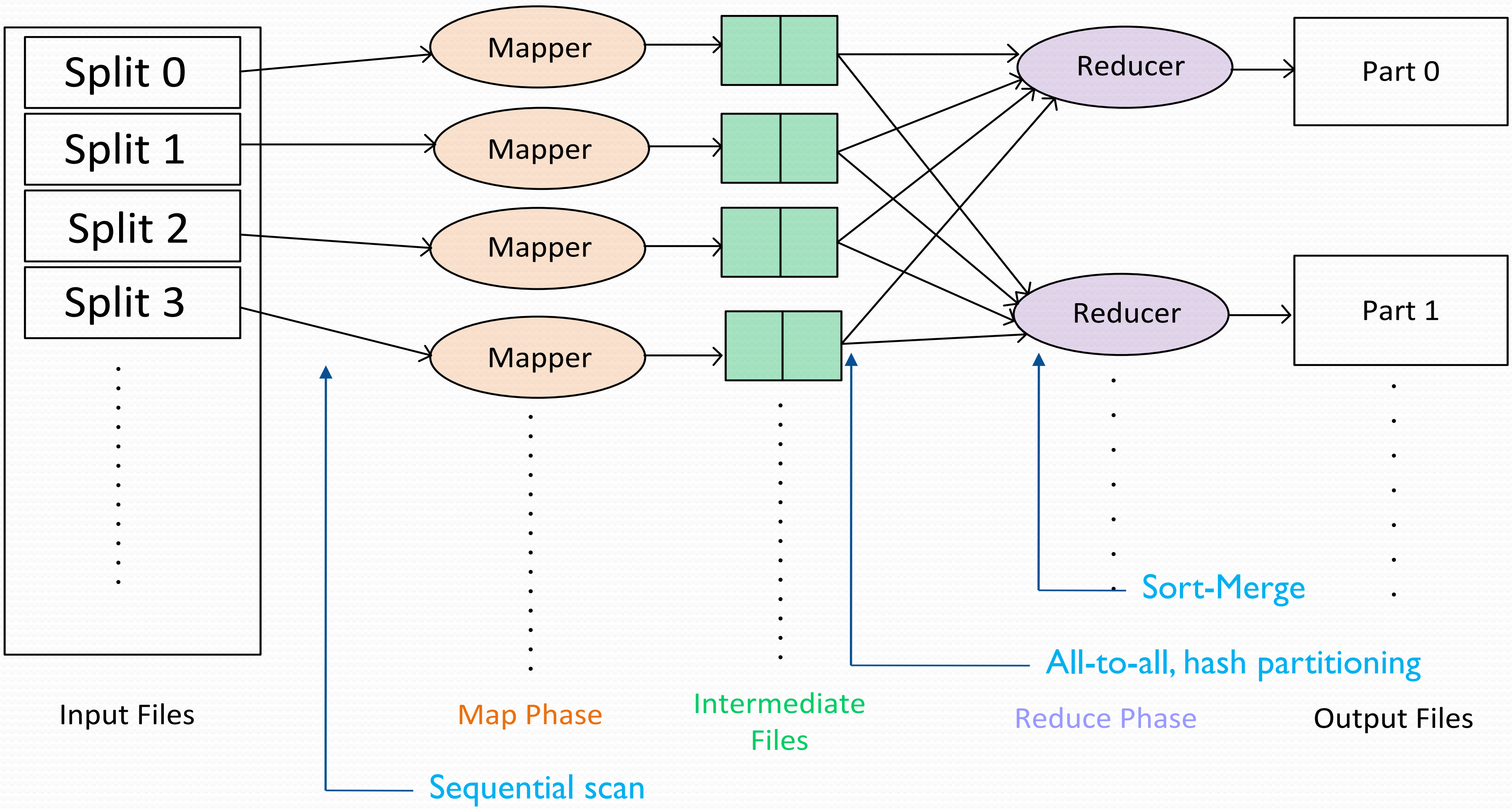
MapReduce: Execution Model



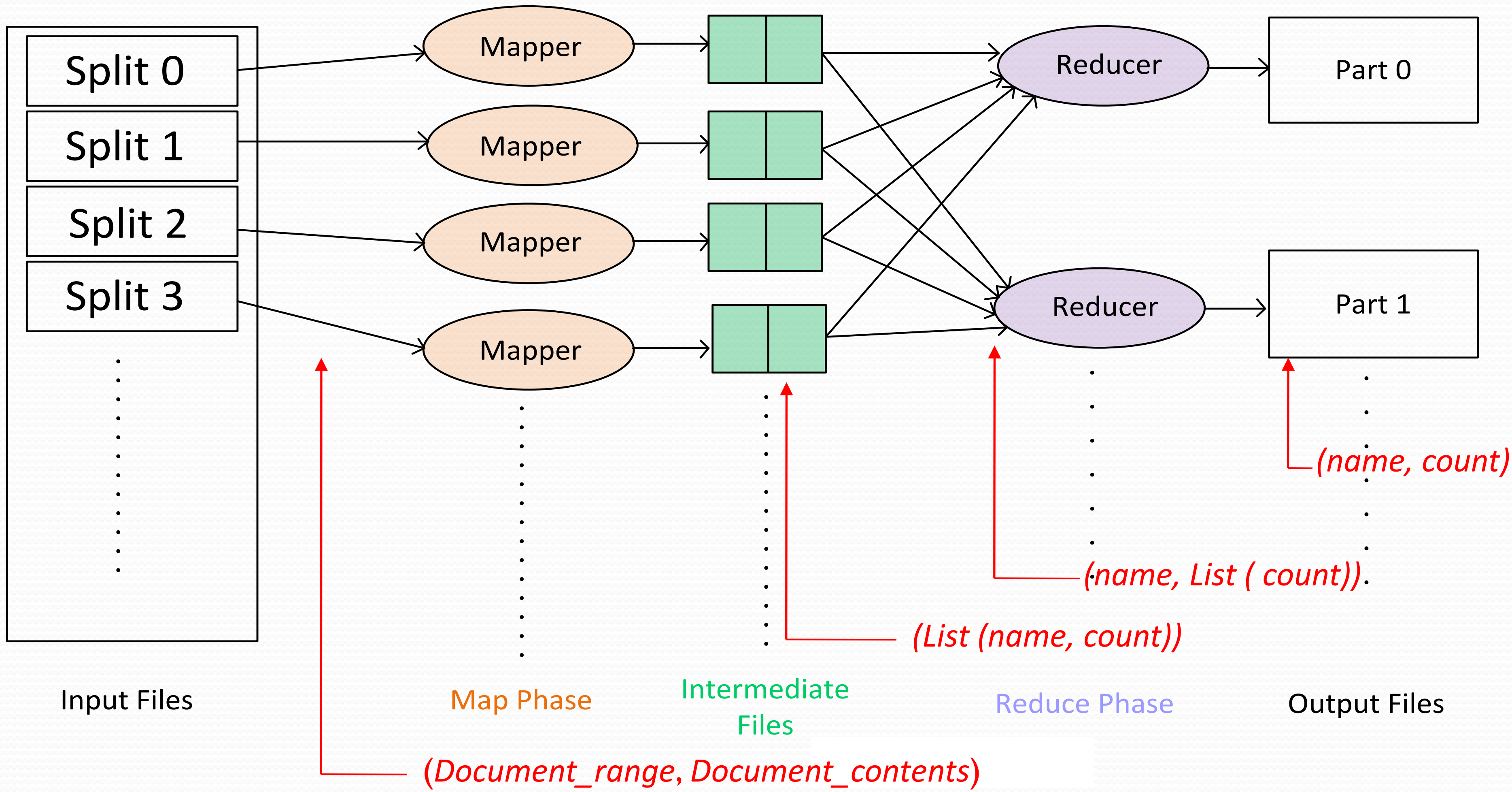
MapReduce: Execution Model - Data Flow



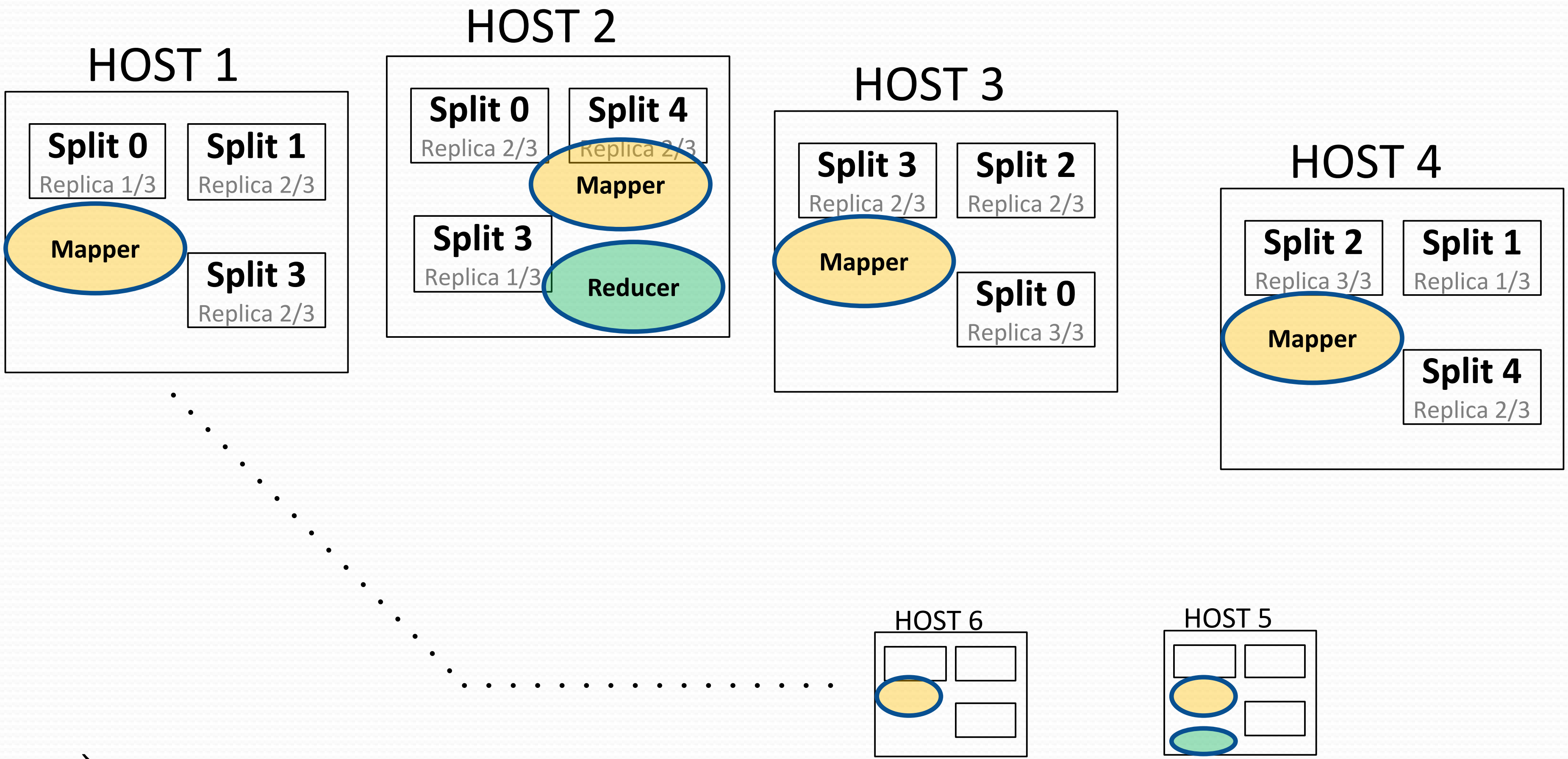
MapReduce: Execution Model - Operations



MapReduce: Execution Model - Types



MapReduce: Execution Model - Placement



⇒ Locality Optimization feature of MapReduce

⇒ Unavoidable Rack/Network traffic

MapReduce: Discussion

- How do different classes of algorithms fit when applying on MapReduce systems?
  1. One iteration algorithms (e.g. single-pass clustering, kNN classification): perfectly fit
  2. Multiple-iteration algorithms (KMeans, Guassian Mixture classifiation): partially fit (some common data has to be shared between iterations)
  3. Multiple-Iteration algorithms with large shared data between iterations (SVM): do not fit
- How about streaming computations?

A Model of *mud* Algorithms (1/5)

- Algorithms written for MapReduce or Hadoop platforms contain massive, unordered, distributed (*mud*) computations*
- *mud* algorithms consist of three functions:
  1. A local function to take a single input data and output a message (applied independently in parallel)
  2. An aggregation function applied to pairs of messages in any order
  3. In some cases: a final post-processing step

* - J. Feldman et. al. On Distributing Symmetric Streaming Computations. In SODA'08

A Model of *mud* Algorithms (2/5)

- An algorithmic model for *mud* algorithms $m = (\Phi, \oplus, \eta)$:
  - $\Phi : \Sigma \to Q$ represents the local function which maps an input item to a message
  - $\oplus : Q \times Q \to Q$ represents the aggregator which maps two messages to a single message
  - $\eta : Q \to \Sigma$ produces the final output

A Model of *mud* Algorithms (4/5)

- An example of a *mud* algorithm $m = (\Phi, \oplus, \eta)$ for calculating the total span of a set of integers:
  - ➔ $\Phi : \Sigma \rightarrow Q; \Phi(x) =< x, x >$
  - ➔ $\oplus : Q \times Q \rightarrow Q;$

    $\oplus(<a_1, b_1>, <a_2, b_2>) =< \min(a_1, a_2), \max(b_1, b_2) >$
  - ➔ $\eta : Q \rightarrow \Sigma; \eta(< a, b >) = b - a$

A Model of *mud* Algorithms (3/5)

- For any binary tree τ with n leaves and for any permutation π of $\{1,\ldots,n\}$, let $m_{\tau,\pi}(X)$ denote the message $q \in Q$ that results from applying $\oplus$ along the topology of τ with the sequence $\Phi(x_1),\ldots,\Phi(x_n)$ with an arbitrary π. The overall output of the mud algorithm is then $\eta(m_{\tau,\pi}(X))$ which is a function $\Sigma^n \to \Sigma$

  - This is to ensure the ability of the mud algorithm to serve as an abstract model of distributed computations that are independent of the underlying implementation

A Model of *mud* Algorithms (5/5)

- Let $q \in Q$, one possible application of $\oplus$ is:

$$\oplus(\oplus(\ldots\oplus(\oplus(q,\Phi(x_1)),\Phi(x_2)),\ldots,\Phi(x_{k-1})),\Phi(x_k))$$

- This sequential application corresponds to the conventional streaming model

Model of Streaming Algorithms

- A streaming algorithm is given by $s = (\sigma, \eta)$ where:
  - $\sigma : Q \times \Sigma \rightarrow Q$ is an operator applied repeatedly to the input stream
  - $\eta : Q \rightarrow \Sigma$ converts the final state to the output

- Let $s^q(X)$ denotes the state of the streaming algorithm after starting at state q, and operating on the sequence $X \in \Sigma^n; X = x_1, \ldots x_n$ exactly in that order such that :

  $s^q(X) = \sigma(\sigma(\ldots\sigma(\sigma(q, x_1), x_2), \ldots x_{n-1}), x_n)$

  Then: the streaming algorithm computes $\eta(s^0(X))$

Streaming Computations vs. MapReduce Computations

- How do mud algorithms and streaming algorithms compare?
  - ➔ Obviously any mud algorithm can be simulated by a stream algorithm in a straightforward way
  - ➔ The question: is it possible to simulate any streaming algorithm using a mud algorithm?

Preliminaries

- We say that a streaming algorithms computes a function f if

$$f : \Sigma^n \to \Sigma; f(X) = \eta(s^0(X))$$

- We say that a function $f : \Sigma^n \to \Sigma$ is computed by a mud algorithm A if $f(X) = \eta(m_{\tau,\pi}(X))$ for all $X \in \Sigma^n$.

Streaming Computations vs. MapReduce Computations

- Theorem*:

For any symmetric function $f : \Sigma^n \rightarrow \Sigma$ computed by a **streaming algorithm** (σ, η) with a $g(n)-$space there exists a **mud algorithm** $(\Phi, \oplus, \eta)$ with a $O(g^2(n))-$space and a comparable communication complexity that also computes f

- Any order-invariant function that can be computed by a streaming algorithm can also be computed by a *mud* algorithm with comparable space and communication complexity

*- J. Feldman et. al. On Distributing Symmetric Streaming Computations. In SODA'08

Streaming Computations vs. MapReduce Computations Summary

- *mud* algorithms are equivalent in power to symmetric streaming algorithms
- For applications on massive data sizes, where even single-pass algorithms are too much: MapReduce-like frameworks are powerful in maintaining parallel single-passes if applied on algorithms which compute symmetric functions
- Recent work on modeling MapReduce: [Karloff et al., SODA 2010]

Summary

- Distributed Stream Processing (DSP) Systems
- Examples on Continuous DSP Systems
- Distributing Computations of Large Data Sets

References (1/2)

1. Cherniack, M., Balakrishnan, H., Balazinska, M., Carney, D., Etintemel, U., Xing, Y., Zdonik S. B.: Scalable Distributed Stream Processing. CIDR 2003.
2. Cormode, G., Muthukrishnan, S., Yi, K.: Algorithms for distributed functional monitoring. In SODA 2008
3. Cormode, G., Muthukrishnan, S., Zhuang, W.: Conquering the divide: Continuous clustering of distributed data streams In ICDE 2007.
4. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Communications of the ACM 51(1): 107-113 (2008)
5. Feldman, J., Muthukrishnan, S., Sidiropoulos, A., Stein, C., Svitkina, Z.: On distributing symmetric streaming computations. In SODA 2008
6. Garofalakis, M. N.: Distributed Data Streams. Pages 883-890 in: Ling Liu, M. Tamer zsu (Eds.): Encyclopedia of Database Systems. 2009
7. Hassani, M., Muller, E., Spaus, P., Faqolli, A., Palpanas, T., Seidl, T.: Self-Organizing Energy Aware Clustering of Nodes in Sensor Networks using Relevant Attributes. In: SensorKDD (2010)

References (2/2)

8. Hassani, M., Muller, E., Seidl, T.: EDISKCO: Energy Efficient Distributed In-Sensor-Network K-center Clustering with Outliers. In: SensorKDD (2009)
9. Karloff, H, Suriy, S. , Vassilvitskii, S.: A Model of Computation for MapReduce . In SODA (2010)
10. Madden S., Franklin M.J., Hellerstein J.M., and HongW. TAG: a tiny aggregation service for ad-hoc sensor networks. USENIX Symp. on Operating System Design and Implementation, (2002)

Thanks for your attention!

Questions?!