cloudera

Common Patterns in Spark Data Processing

Chapter 16

Course Chapters

Chapter	Title	Section
1	Introduction	Course Introduction
2	Introduction to Hadoop and the Hadoop Ecosystem	Introduction to Hadoop
3	Hadoop Architecture and HDFS	Introduction to Hadoop
4	Importing Relational Data with Apache Sqoop	Importing and Modeling Structured Data
5	Introduction to Impala and Hive	Importing and Modeling Structured Data
6	Modeling and Managing Data with Impala and Hive	Importing and Modeling Structured Data
7	Data Formats	Importing and Modeling Structured Data
8	Data File Partitioning	Importing and Modeling Structured Data
9	Capturing Data with Apache Flume	Ingesting Streaming Data
10	Spark Basics	**Distributed Data Processing with Spark**
11	Working with RDDs in Spark	**Distributed Data Processing with Spark**
12	Aggregating Data with Pair RDDs	**Distributed Data Processing with Spark**
13	Writing and Deploying Spark Applications	**Distributed Data Processing with Spark**
14	Parallel Processing in Spark	**Distributed Data Processing with Spark**
15	Spark RDD Persistence	**Distributed Data Processing with Spark**
16	**Common Patterns in Spark Data Processing**	**Distributed Data Processing with Spark**
17	Spark SQL and DataFrames	**Distributed Data Processing with Spark**
18	Conclusion	Course Conclusion

Common Patterns in Spark Programming

In this chapter you will learn

- **What kinds of processing and analysis Spark is best at**
- **How to implement an iterative algorithm in Spark**
- **How GraphX and MLlib work with Spark**

Chapter Topics

Common Patterns in Spark Data Processing	Distributed Data Processing with Spark

- **Common Spark Use Cases**
- Iterative Algorithms in Spark
- Graph Processing and Analysis
- Machine Learning
- Example: k-means
- Conclusion
- Homework: Implement an Iterative Algorithm with Spark
- Optional Homework: Partition Data Files Using Spark

Common Spark Use Cases (1)

- **Spark is especially useful when working with any combination of:**
  - Large amounts of data
    - Distributed storage
  - Intensive computations
    - Distributed computing
  - Iterative algorithms
    - In-memory processing and pipelining

Common Spark Use Cases (2)

- **Examples**
  - Risk analysis
    - "How likely is this borrower to pay back a loan?"
  - Recommendations
    - "Which products will this customer enjoy?"
  - Predictions
    - "How can we prevent service outages instead of simply reacting to them?"
  - Classification
    - "How can we tell which mail is spam and which is legitimate?"

Spark Examples

- **Spark includes many example programs that demonstrate some common Spark programming patterns and algorithms**
  - k-means
  - Logistic regression
  - Calculate pi
  - Alternating least squares (ALS)
  - Querying Apache web logs
  - Processing Twitter feeds
- **Examples**
  - **`$SPARK_HOME/examples/lib`**
    - **`spark-examples-`*`version`*`.jar` – Java and Scala examples**
    - **`python.tar.gz` – Pyspark examples**

Chapter Topics

Common Patterns in Spark Data Processing	Distributed Data Processing with Spark

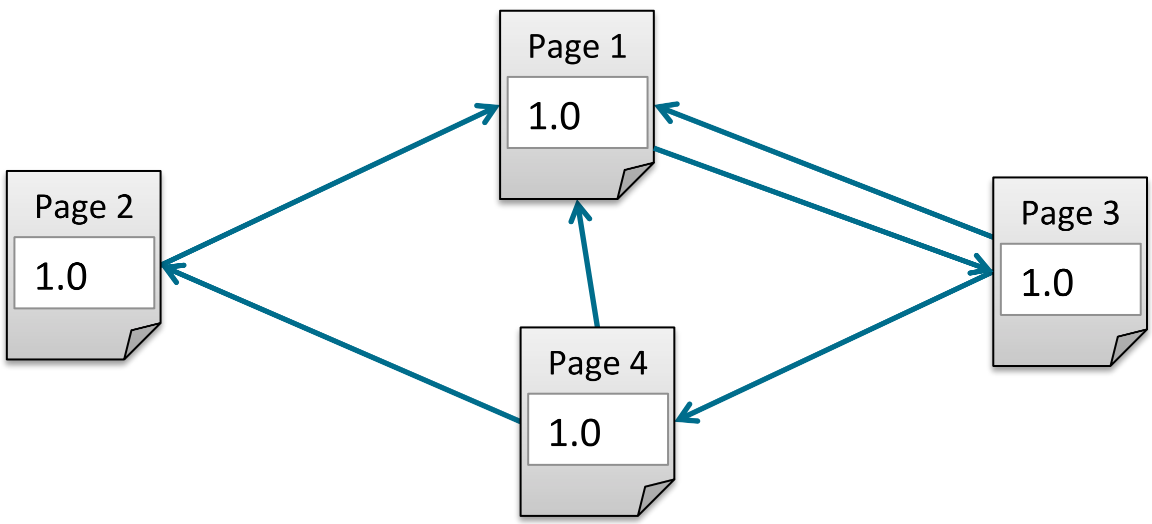
- Common Spark Use Cases
- **Iterative Algorithms in Spark**
- Graph Processing and Analysis
- Machine Learning
- Example: k-means
- Conclusion
- Homework: Implement an Iterative Algorithm with Spark
- Optional Homework: Partition Data Files Using Spark

Example: PageRank

- **PageRank gives web pages a ranking score based on links from other pages**
  - Higher scores given for more links, and links from other high ranking pages
- **Why do we care?**
  - PageRank is a classic example of big data analysis (like WordCount)
    - Lots of data – needs an algorithm that is distributable and scalable
    - Iterative – the more iterations, the better than answer

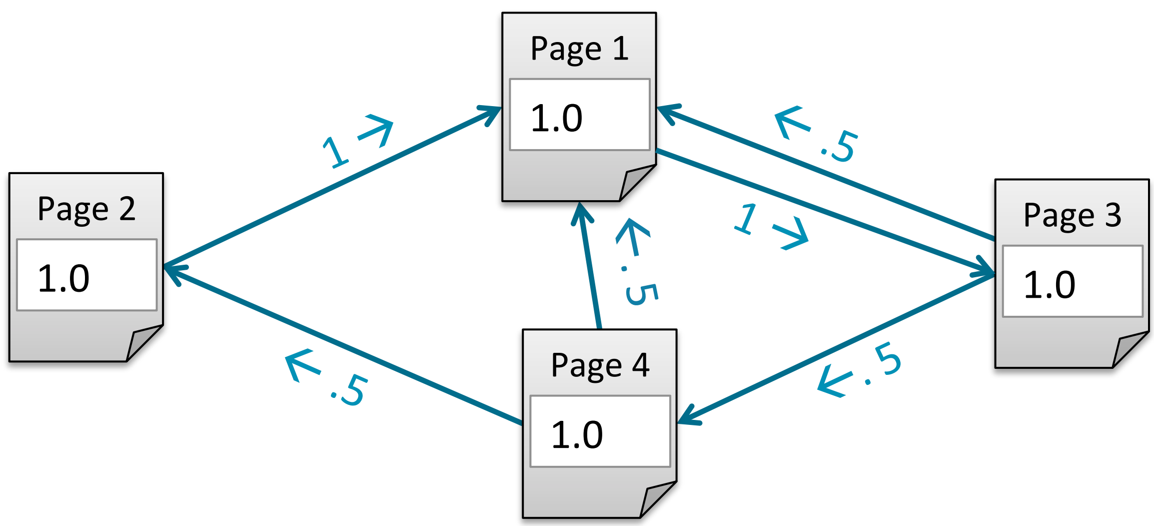
PageRank Algorithm (1)

1. **Start each page with a rank of 1.0**



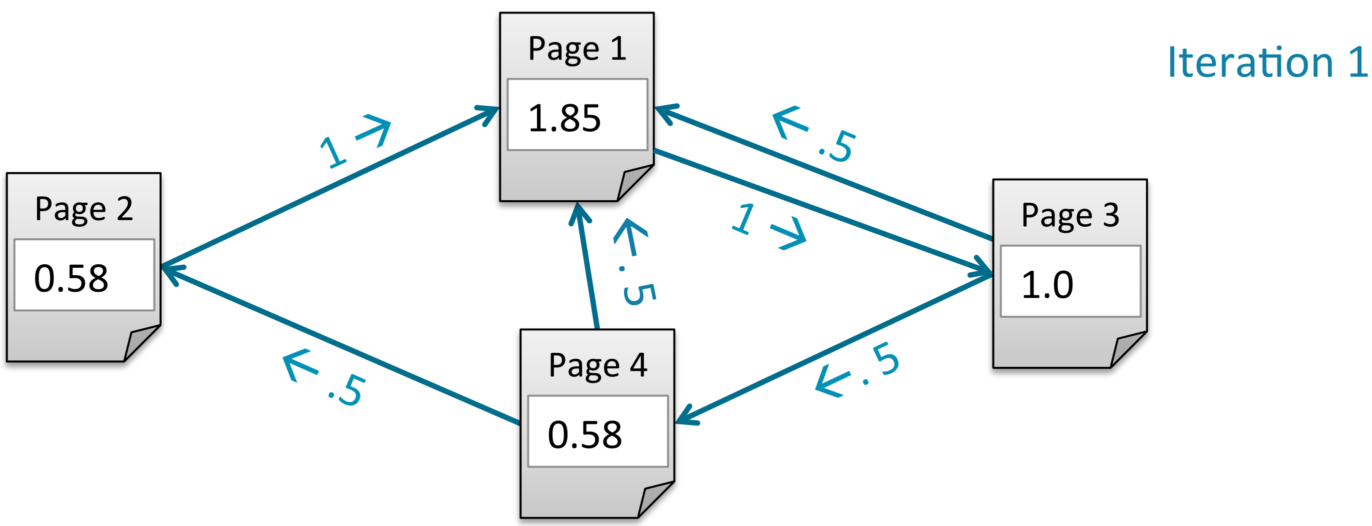
PageRank Algorithm (2)

1. **Start each page with a rank of 1.0**
2. **On each iteration:**
   1. each page contributes to its neighbors its own rank divided by the number of its neighbors: $contrib_p = rank_p / neighbors_p$



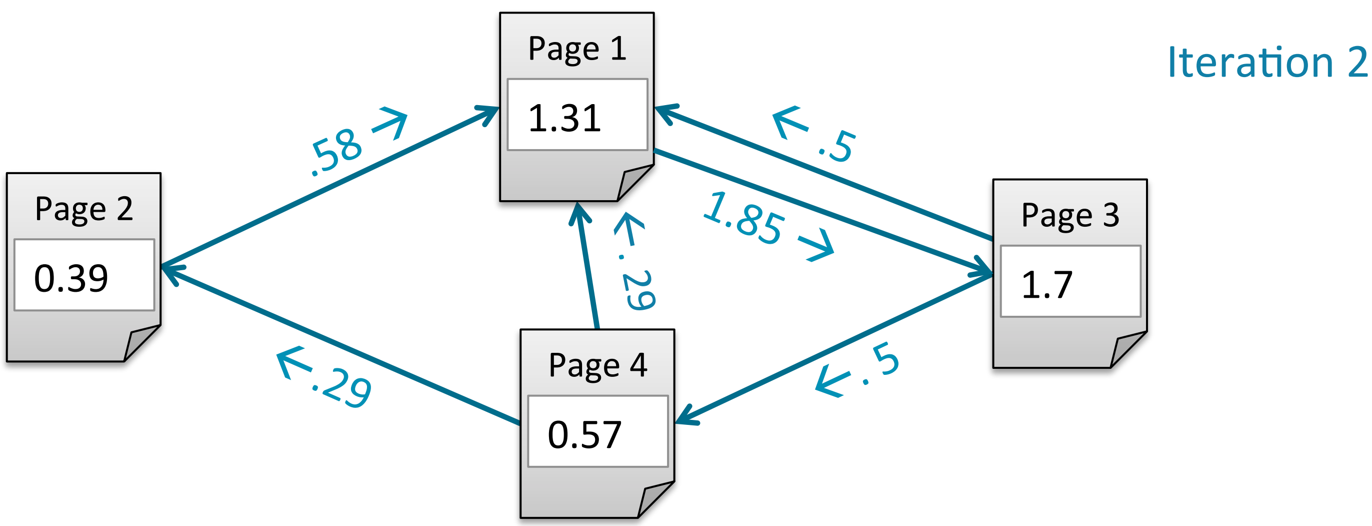
PageRank Algorithm (3)

1. **Start each page with a rank of 1.0**
2. **On each iteration:**
   1. each page contributes to its neighbors its own rank divided by the number of its neighbors: $contrib_p = rank_p / neighbors_p$
   2. Set each page's new rank based on the sum of its neighbors contribution: $new\text{-}rank = \Sigma contribs * .85 + .15$



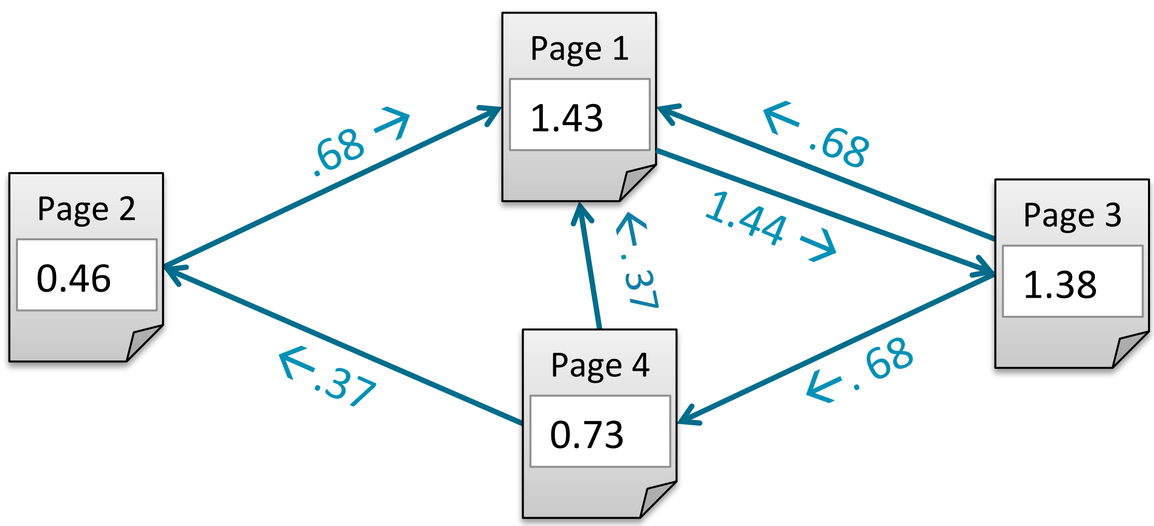
PageRank Algorithm (4)

1. **Start each page with a rank of 1.0**
2. **On each iteration:**
    1. each page contributes to its neighbors its own rank divided by the number of its neighbors: $contrib_p = rank_p / neighbors_p$
    2. Set each page's new rank based on the sum of its neighbors contribution: $new\text{-}rank = \Sigma contribs * .85 + .15$
3. **Each iteration incrementally improves the page ranking**



PageRank Algorithm (5)

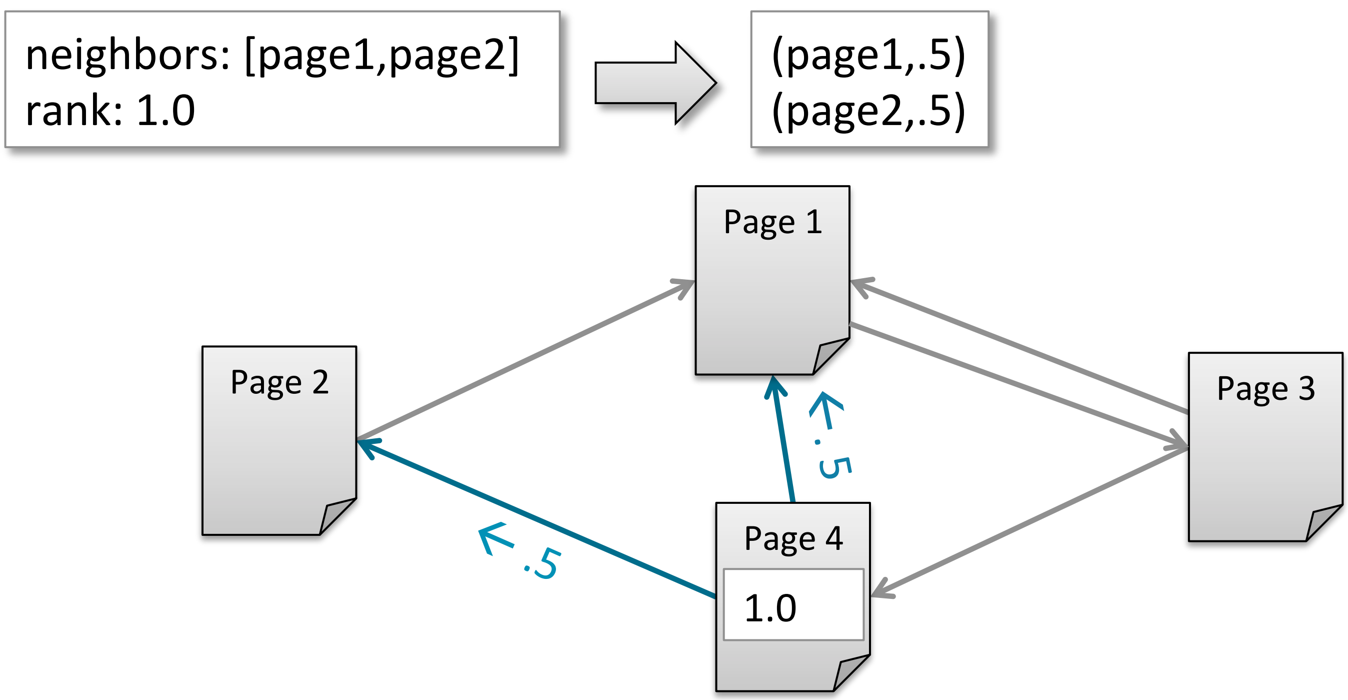
1. **Start each page with a rank of 1.0**
2. **On each iteration:**
   1. each page contributes to its neighbors its own rank divided by the number of its neighbors: $contrib_p = rank_p / neighbors_p$
   2. Set each page's new rank based on the sum of its neighbors contribution: $new\text{-}rank = \Sigma contribs * .85 + .15$
3. **Each iteration incrementally improves the page ranking**



Iteration 10
(Final)

PageRank in Spark: Neighbor Contribution Function

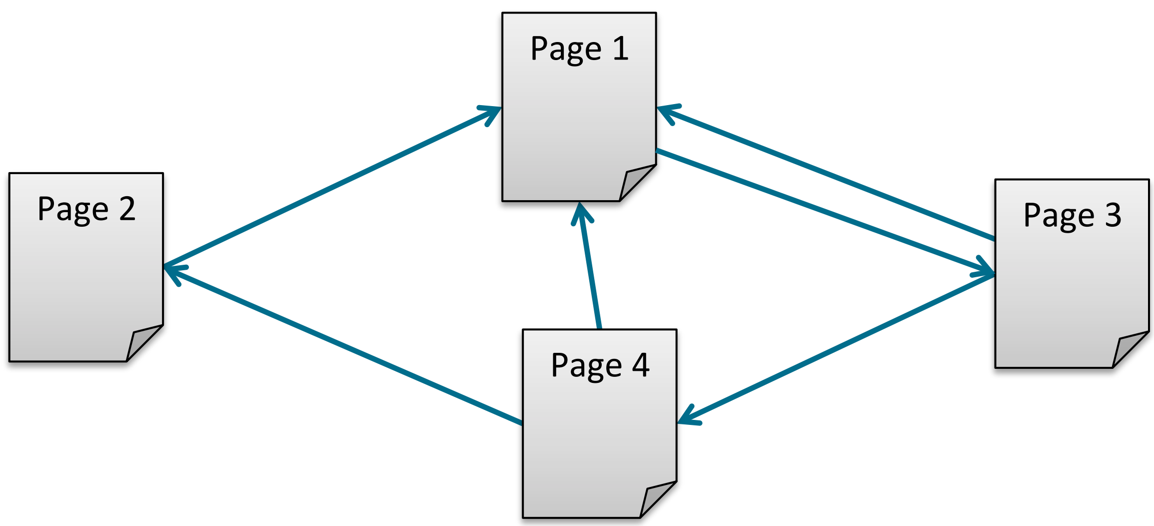
```
def computeContribs(neighbors, rank):
    for neighbor in neighbors: yield(neighbor, rank/len(neighbors))
```



PageRank in Spark: Example Data

Data Format:
source-page destination-page
…

```
page1 page3
page2 page1
page4 page1
page3 page1
page4 page2
page3 page4
```



PageRank in Spark: Pairs of Page Links

```
def computeContribs(neighbors, rank):…

links = sc.textFile(file)\
   .map(lambda line: line.split())\
   .map(lambda pages: (pages[0],pages[1]))\
   .distinct()
```

```
page1 page3
page2 page1
page4 page1
page3 page1
page4 page2
page3 page4
```

(page1,page3)
(page2,page1)
(page4,page1)
(page3,page1)
(page4,page2)
(page3,page4)

PageRank in Spark: Page Links Grouped by Source Page

```
def computeContribs(neighbors, rank):…

links = sc.textFile(file)\
    .map(lambda line: line.split())\
    .map(lambda pages: (pages[0],pages[1]))\
    .distinct()\
    .groupByKey()
```

```
page1 page3
page2 page1
page4 page1
page3 page1
page4 page2
page3 page4
```

(page1,page3)
(page2,page1)
(page4,page1)
(page3,page1)
(page4,page2)
(page3,page4)

links

(page4, [page2,page1])
(page2, [page1])
(page3, [page1,page4])
(page1, [page3])

PageRank in Spark: Persisting the Link Pair RDD

```
def computeContribs(neighbors, rank):…

links = sc.textFile(file)\
    .map(lambda line: line.split())\
    .map(lambda pages: (pages[0],pages[1]))\
    .distinct()\
    .groupByKey()\
    .persist()
```

```
page1 page3
page2 page1
page4 page1
page3 page1
page4 page2
page3 page4
```

```
(page1,page3)
(page2,page1)
(page4,page1)
(page3,page1)
(page4,page2)
(page3,page4)
```

links

```
(page4, [page2,page1])
(page2, [page1])
(page3, [page1,page4])
(page1, [page3])
```

PageRank in Spark: Set Initial Ranks

```
def computeContribs(neighbors, rank):…

links = sc.textFile(file)\
   .map(lambda line: line.split())\
   .map(lambda pages: (pages[0],pages[1]))\
   .distinct()\
   .groupByKey()\
   .persist()

ranks=links.map(lambda (page,neighbors): (page,1.0))
```

links
(page4, [page2,page1])
(page2, [page1])
(page3, [page1,page4])
(page1, [page3])

ranks
(page4, 1.0)
(page2, 1.0)
(page3, 1.0)
(page1, 1.0)

PageRank in Spark: First Iteration (1)

```
def computeContribs(neighbors, rank):…

links = …

ranks = …

for x in xrange(10):
  contribs=links\
    .join(ranks)
```

links

(page4, [page2,page1])
(page2, [page1])
(page3, [page1,page4])
(page1, [page3])

ranks

(page4, 1.0)
(page2, 1.0)
(page3, 1.0)
(page1, 1.0)

(page4, ([page2,page1], 1.0))
(page2, ([page1], 1.0))
(page3, ([page1,page4], 1.0))
(page1, ([page3], 1.0))

PageRank in Spark: First Iteration (2)

```
def computeContribs(neighbors, rank):…

links = …

ranks = …

for x in xrange(10):
  contribs=links\
    .join(ranks)\
    .flatMap(lambda (page,(neighbors,rank)): \
      computeContribs(neighbors,rank))
```

links

(page4, [page2,page1])
(page2, [page1])
(page3, [page1,page4])
(page1, [page3])

ranks

(page4, 1.0)
(page2, 1.0)
(page3, 1.0)
(page1, 1.0)

(page4, ([page2,page1], 1.0))
(page2, ([page1], 1.0))
(page3, ([page1,page4], 1.0))
(page1, ([page3], 1.0))

contribs

(page2,0.5)
(page1,0.5)
(page1,1.0)
(page1,0.5)
(page4,0.5)
(page3,1.0)

PageRank in Spark: First Iteration (3)

```
def computeContribs(neighbors, rank):…

links = …

ranks = …

for x in xrange(10):
  contribs=links\
    .join(ranks)\
    .flatMap(lambda (page,(neighbors,rank)): \
       computeContribs(neighbors,rank))
  ranks=contribs\
    .reduceByKey(lambda v1,v2: v1+v2)
```

contribs
(page2,0.5)
(page1,0.5)
(page1,1.0)
(page1,0.5)
(page4,0.5)
(page3,1.0)

(page4,0.5)
(page2,0.5)
(page3,1.0)
(page1,2.0)

PageRank in Spark: First Iteration (4)

```
def computeContribs(neighbors, rank):…

links = …

ranks = …

for x in xrange(10):
  contribs=links\
    .join(ranks)\
    .flatMap(lambda (page,(neighbors,rank)): \
       computeContribs(neighbors,rank))
  ranks=contribs\
    .reduceByKey(lambda v1,v2: v1+v2)\
    .map(lambda (page,contrib): \
         (page,contrib * 0.85 + 0.15))
```

contribs
(page2,0.5)
(page1,0.5)
(page1,1.0)
(page1,0.5)
(page4,0.5)
(page3,1.0)

(page4,0.5)
(page2,0.5)
(page3,1.0)
(page1,2.0)

ranks
(page4,.58)
(page2,.58)
(page3,1.0)
(page1,1.85)

PageRank in Spark: Second Iteration

```
def computeContribs(neighbors, rank):…

links = …

ranks = …

for x in xrange(10):
  contribs=links\
    .join(ranks)\
    .flatMap(lambda (page,(neighbors,rank)): \
       computeContribs(neighbors,rank))
  ranks=contribs\
    .reduceByKey(lambda v1,v2: v1+v2)\
    .map(lambda (page,contrib): \
         (page,contrib * 0.85 + 0.15))

for rank in ranks.collect(): print rank
```

links

(page4, [page2,page1])
(page2, [page1])
(page3, [page1,page4])
(page1, [page3])

ranks

(page4,0.58)
(page2,0.58)
(page3,1.0)
(page1,1.85)

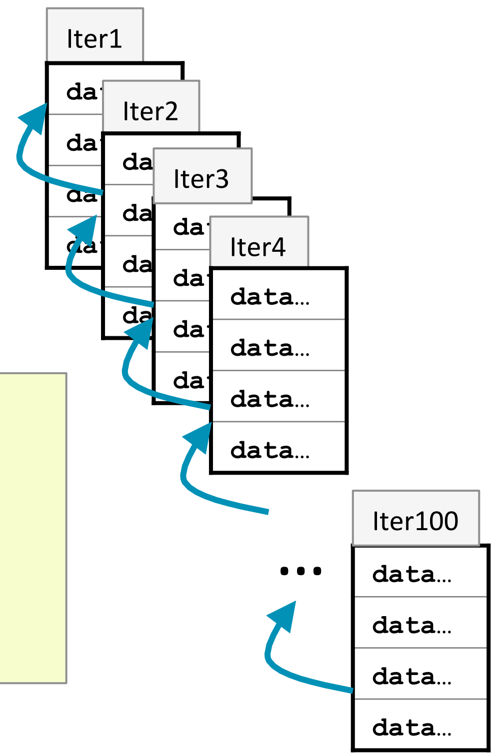
…

ranks

(page4,0.57)
(page2,0.21)
(page3,1.0)
(page1,0.77)

Checkpointing (1)

- **Maintaining RDD lineage provides resilience but can also cause problems when the lineage gets very long**
  - e.g., iterative algorithms, streaming
- **Recovery can be very expensive**
- **Potential stack overflow**

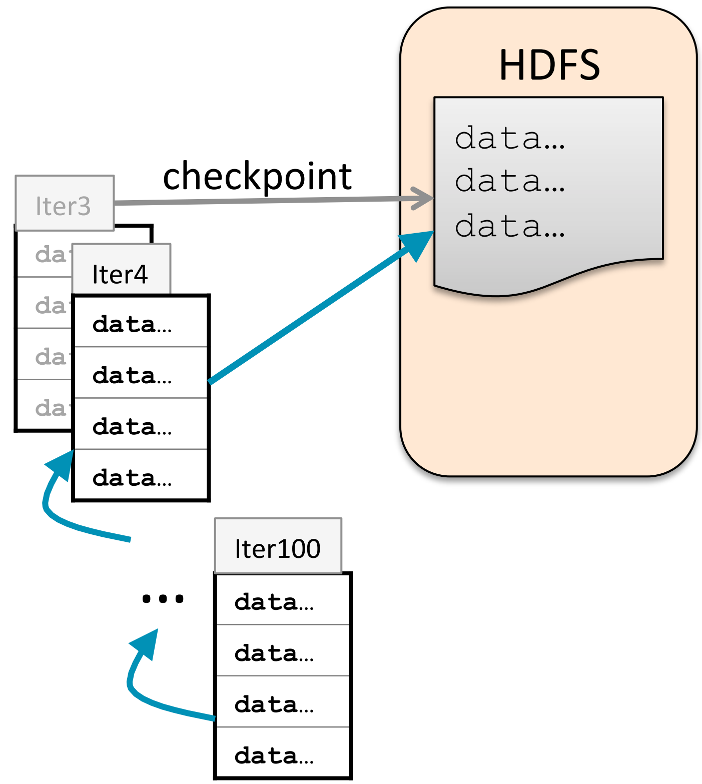



```
myrdd = …initial-value…
while x in xrange(100):
   myrdd = myrdd.transform(…)
myrdd.saveAsTextFile(dir)
```

Checkpointing (2)

- **Checkpointing saves the data to HDFS**
  - Provides fault-tolerant storage across nodes
- **Lineage is not saved**
- **Must be checkpointed before any actions on the RDD**

```
sc.setCheckpointDir(directory)
myrdd = …initial-value…
while x in xrange(100):
  myrdd = myrdd.transform(…)
  if x % 3 == 0:
    myrdd.checkpoint()
    myrdd.count()
myrdd.saveAsTextFile(dir)
```



Chapter Topics

Common Patterns in Spark Data Processing	Distributed Data Processing with Spark

- Common Spark Use Cases
- Iterative Algorithms in Spark
- **Graph Processing and Analysis**
- Machine Learning
- Example: k-means
- Conclusion
- Homework: Implement an Iterative Algorithm with Spark
- Optional Homework: Partition Data Files Using Spark

Graph Analytics

- **Many data analytics problems work with "data parallel" algorithms**
  - Records can be processed independently of each other
  - Very well suited to parallelizing
- **Some problems focus on the relationships between the individual data items. For example:**
  - Social networks
  - Web page hyperlinks
  - Roadmaps
- **These relationships can be represented by graphs**
  - Requires "graph parallel" algorithms

Graph Analysis Challenges at Scale

- **Graph Creation**
  - Extracting relationship information from a data source
    - For example, extracting links from web pages
- **Graph Representation**
  - e.g., adjacency lists in a table
- **Graph Analysis**
  - Inherently iterative, hard to parallelize
  - This is the focus of specialized libraries like Pregel, GraphLab
- **Post-analysis processing**
  - e.g., incorporating product recommendations into a retail site

Graph Analysis in Spark

- **Spark is very well suited to graph parallel algorithms**
- **GraphX**
  - UC Berkeley AMPLab project on top of Spark
  - Unifies optimized graph computation with Spark's fast data parallelism and interactive abilities
  - Supersedes predecessor Bagel (Pregel on Spark)

Chapter Topics

Common Patterns in Spark Data Processing	Distributed Data Processing with Spark

- Common Spark Use Cases
- Iterative Algorithms in Spark
- Graph Processing and Analysis
- **Machine Learning**
- Example: k-means
- Conclusion
- Homework: Implement an Iterative Algorithm with Spark
- Optional Homework: Partition Data Files Using Spark

Machine Learning

- **Most programs tell computers exactly what to do**
  - Database transactions and queries
  - Controllers
    - Phone systems, manufacturing processes, transport, weaponry, etc.
  - Media delivery
  - Simple search
  - Social systems
    - Chat, blogs, email, etc.
- **An alternative technique is to have computers *learn* what to do**
- **Machine Learning refers to programs that leverage collected data to drive future program behavior**
- **This represents another major opportunity to gain value from data**

The 'Three Cs'

- **Machine Learning is an active area of research and new applications**
- **There are three well-established categories of techniques for exploiting data**
  - Collaborative filtering (recommendations)
  - Clustering
  - Classification

Collaborative Filtering

- **Collaborative Filtering is a technique for recommendations**
- **Example application: given people who each like certain books, learn to suggest what someone may like in the future based on what they already like**
- **Helps users navigate data by expanding to topics that have affinity with their established interests**
- **Collaborative Filtering algorithms are agnostic to the different types of data items involved**
  - Useful in many different domains

Clustering

- **Clustering algorithms discover structure in collections of data**
  - Where no formal structure previously existed
- **They discover what clusters, or groupings, naturally occur in data**
- **Examples**
  - Finding related news articles
  - Computer vision (groups of pixels that cohere into objects)

Classification

- **The previous two techniques are considered 'unsupervised' learning**
  - The algorithm discovers groups or recommendations itself
- **Classification is a form of 'supervised' learning**
- **A classification system takes a set of data records with known labels**
  - Learns how to label new records based on that information
- **Examples**
  - Given a set of e-mails identified as spam/not spam, label new e-mails as spam/not spam
  - Given tumors identified as benign or malignant, classify new tumors

Machine Learning Challenges

- **Highly computation intensive and iterative**
- **Many traditional numerical processing systems do not scale to very large datasets**
  - e.g., MatLab

MLlib: Machine Learning on Spark

- **MLlib is part of Apache Spark**
- **Includes many common ML functions**
  - ALS (alternating least squares)
  - k-means
  - Logistic Regression
  - Linear Regression
  - Gradient Descent
- **Still a 'work in progress'**

Chapter Topics

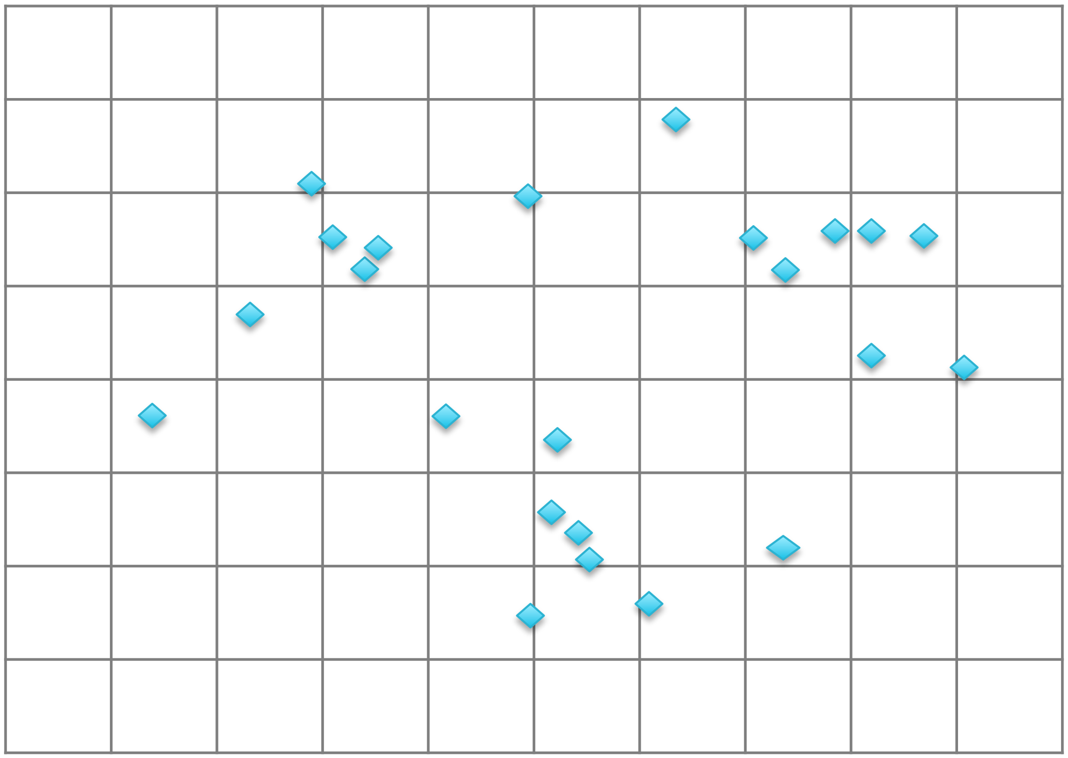
Common Patterns in Spark Data Processing	Distributed Data Processing with Spark

- Common Spark Use Cases
- Iterative Algorithms in Spark
- Graph Processing and Analysis
- Machine Learning
- **Example: k-means**
- Conclusion
- Homework: Implement an Iterative Algorithm with Spark
- Optional Homework: Partition Data Files Using Spark

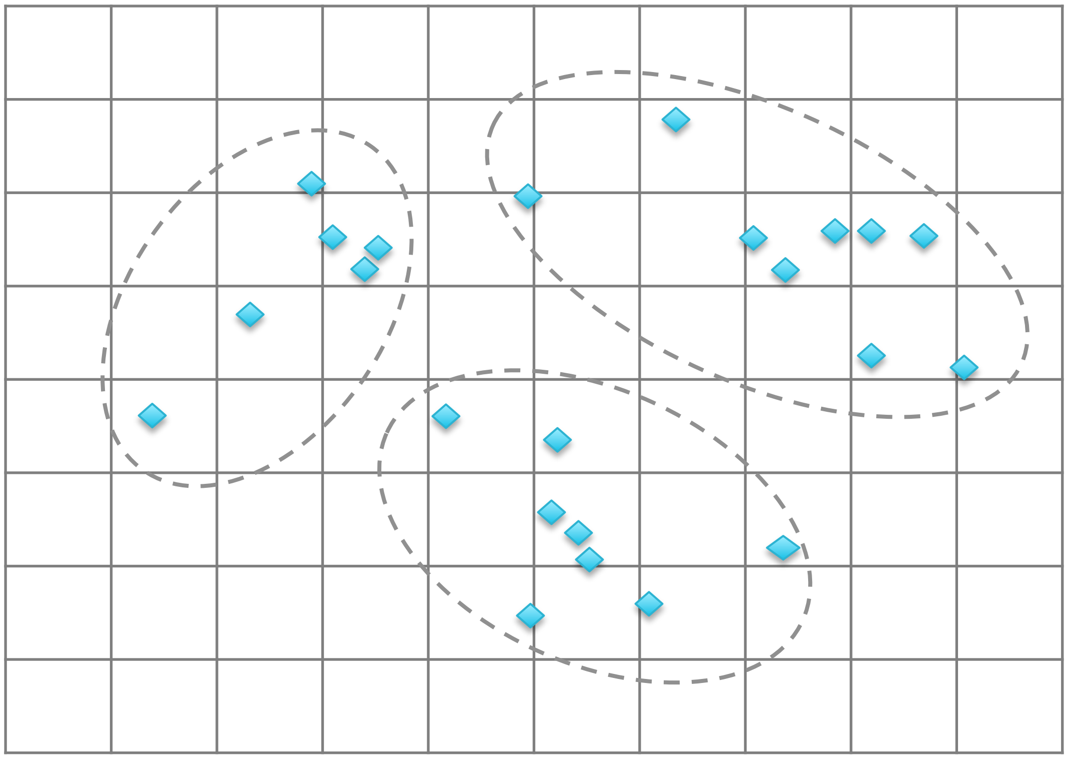
k-means Clustering

- **k-means Clustering**
  - A common iterative algorithm used in graph analysis and machine learning
  - You will implement a simplified version in the homework assignment

Clustering (1)

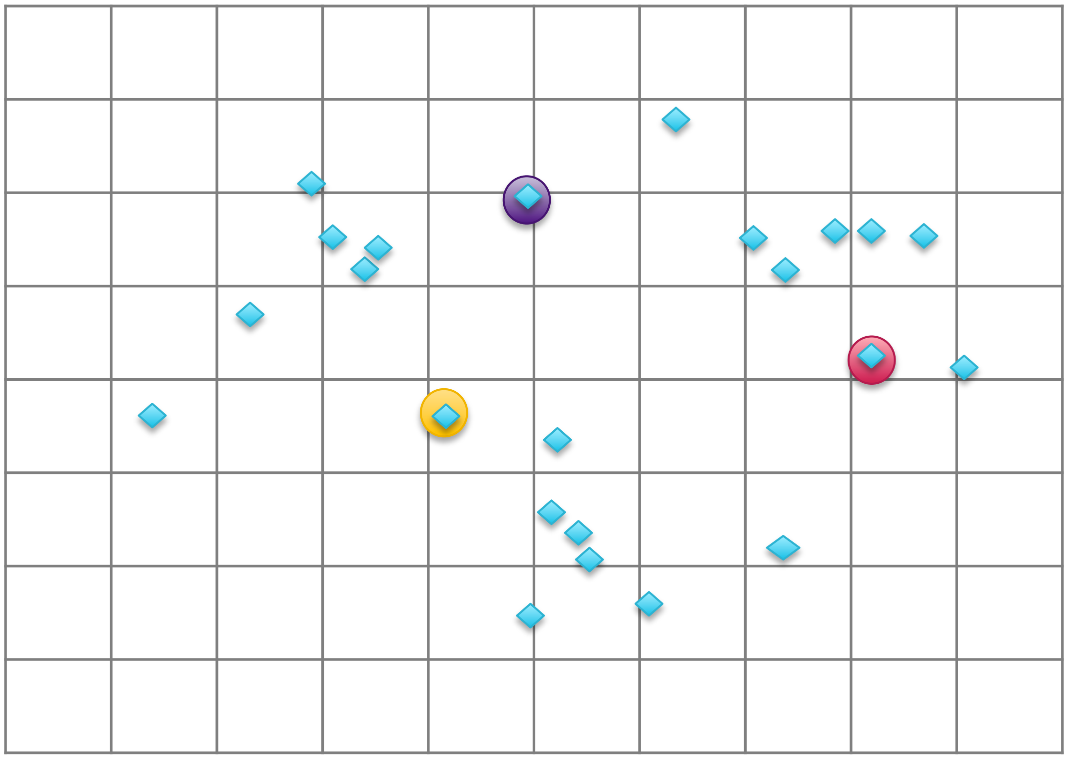


Clustering (2)



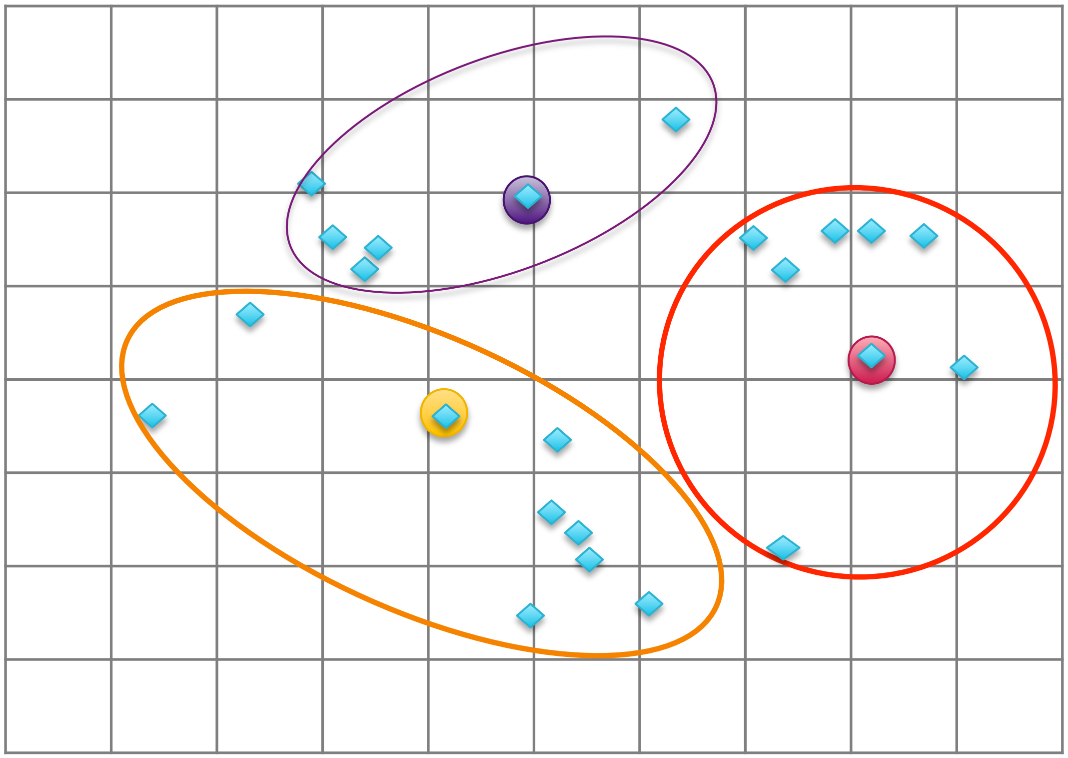
Goal: Find "clusters" of data points

Example: k-means Clustering (1)



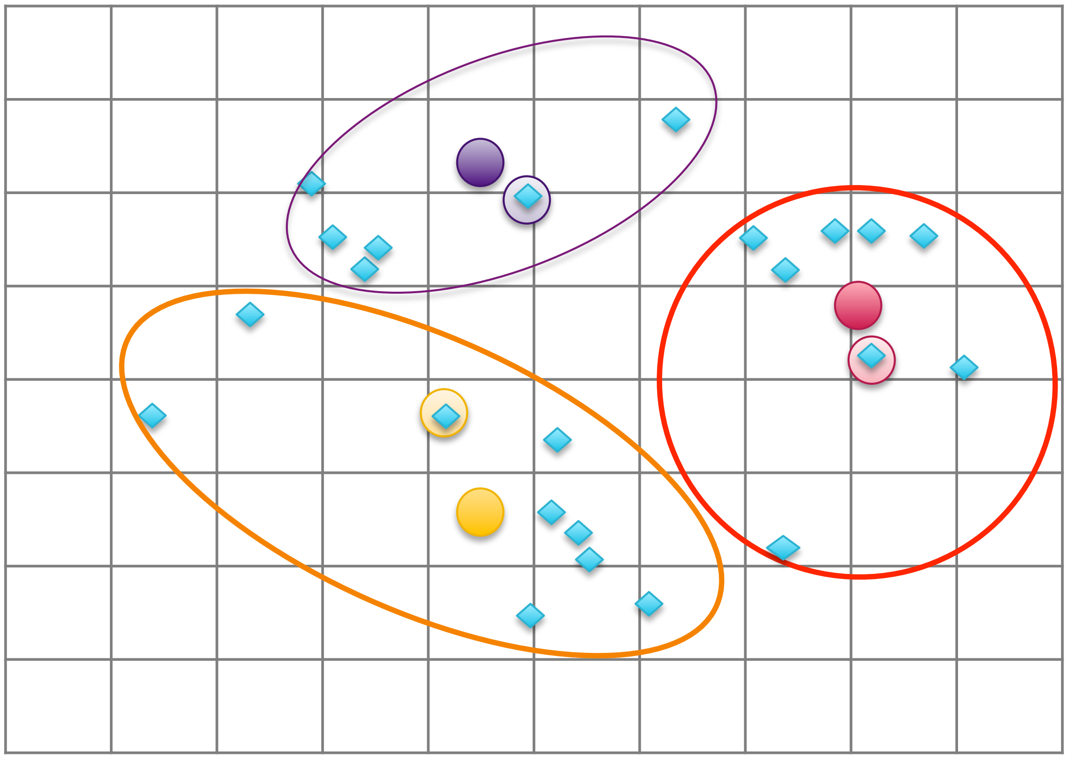
1. Choose *K* random points as starting centers

Example: k-means Clustering (2)



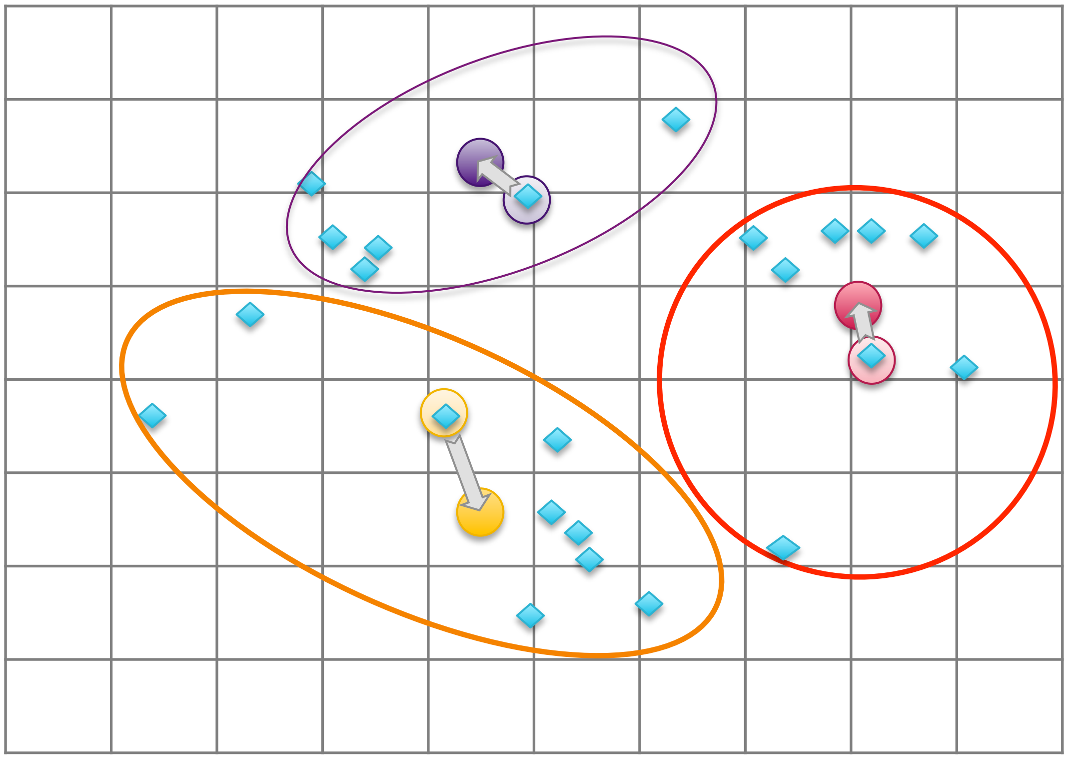
1. Choose *K* random points as starting centers
2. Find all points closest to each center

Example: k-means Clustering (3)



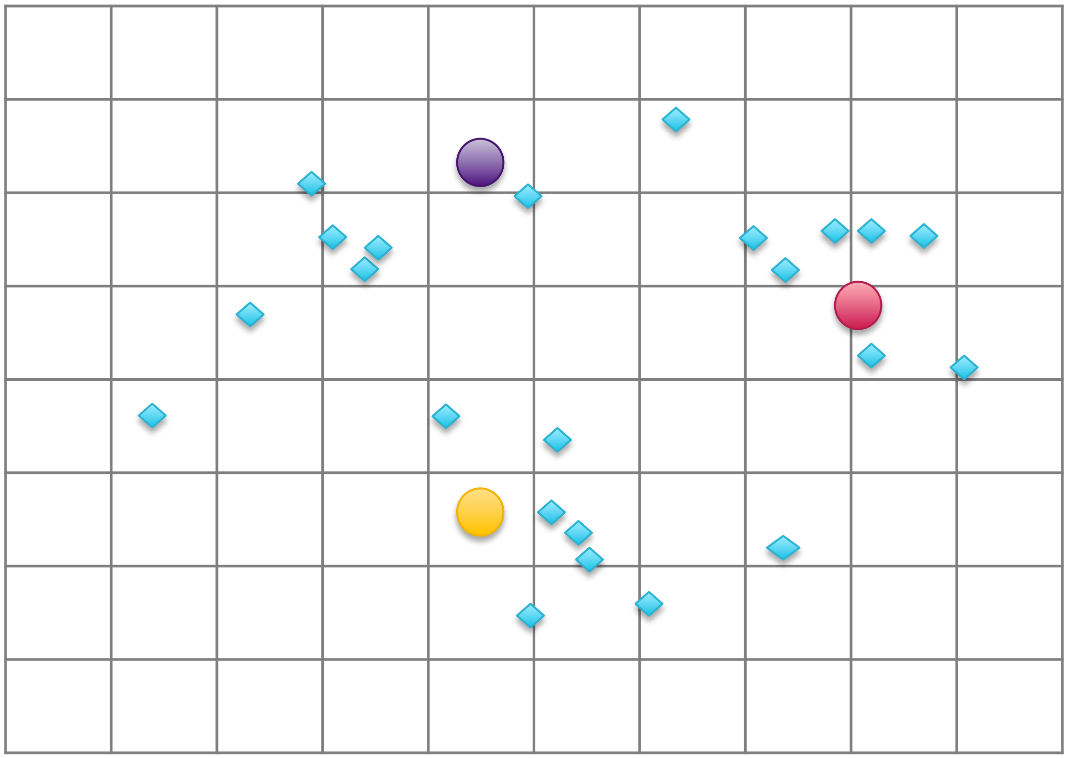
1. Choose *K* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster

Example: k-means Clustering (4)



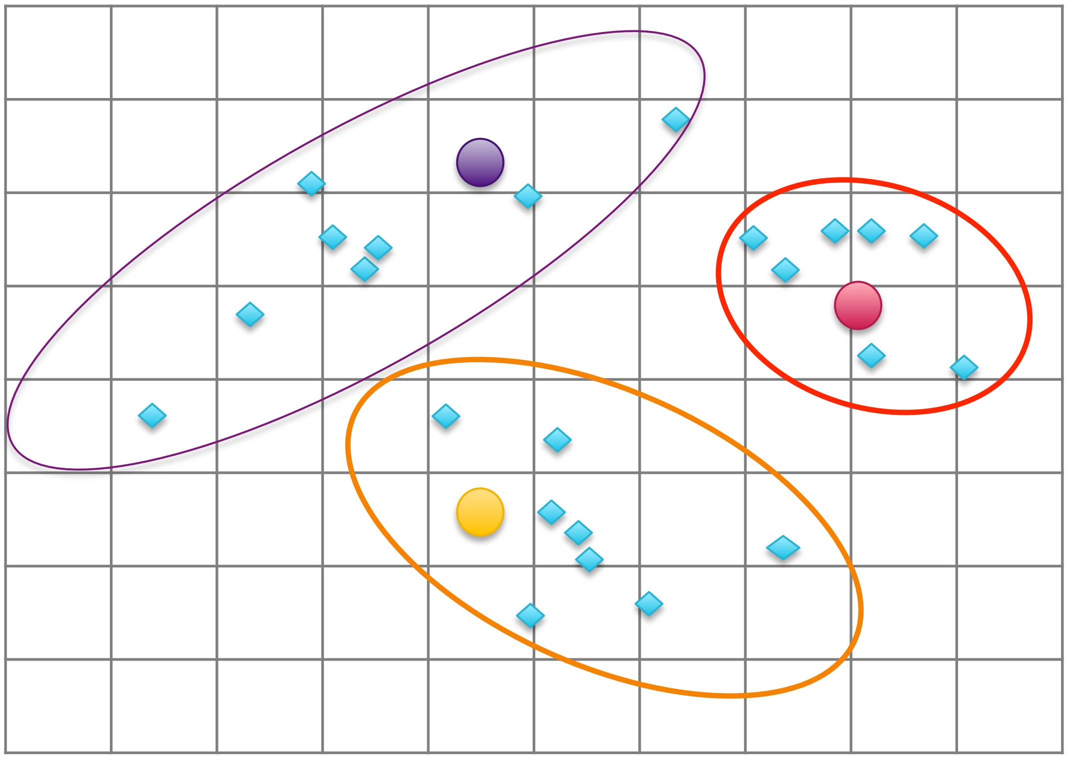
1. Choose *K* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed, iterate again

Example: k-means Clustering (5)



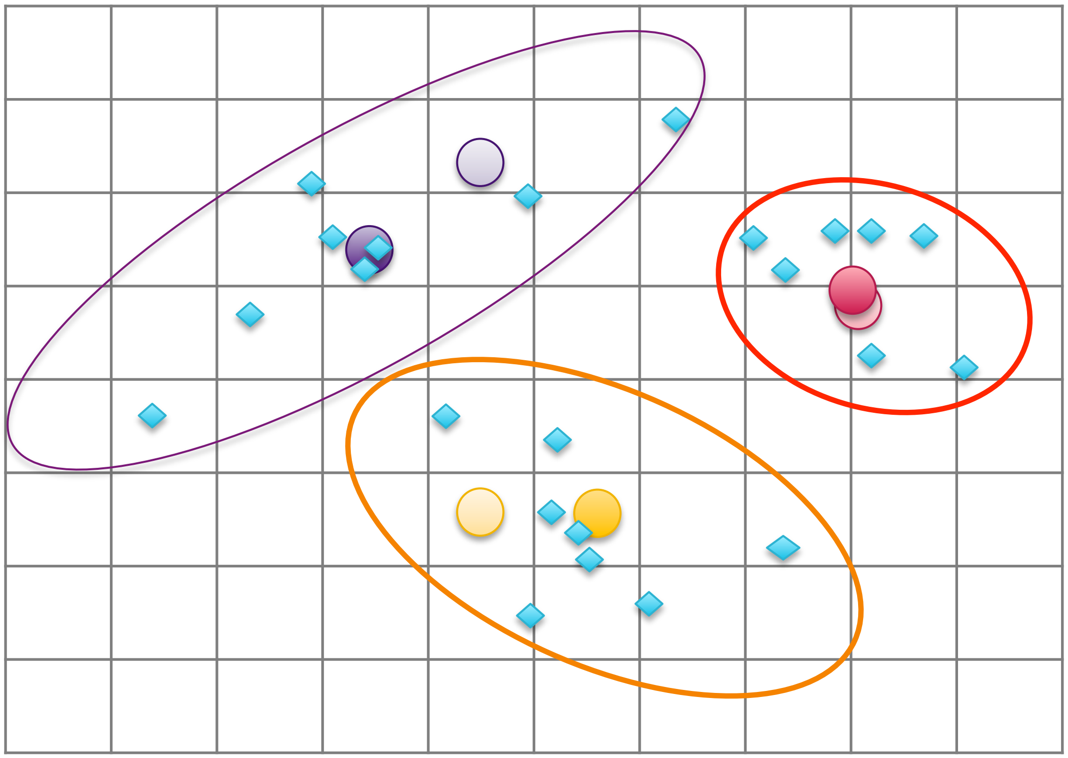
1. Choose *K* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed, iterate again

Example: k-means Clustering (6)



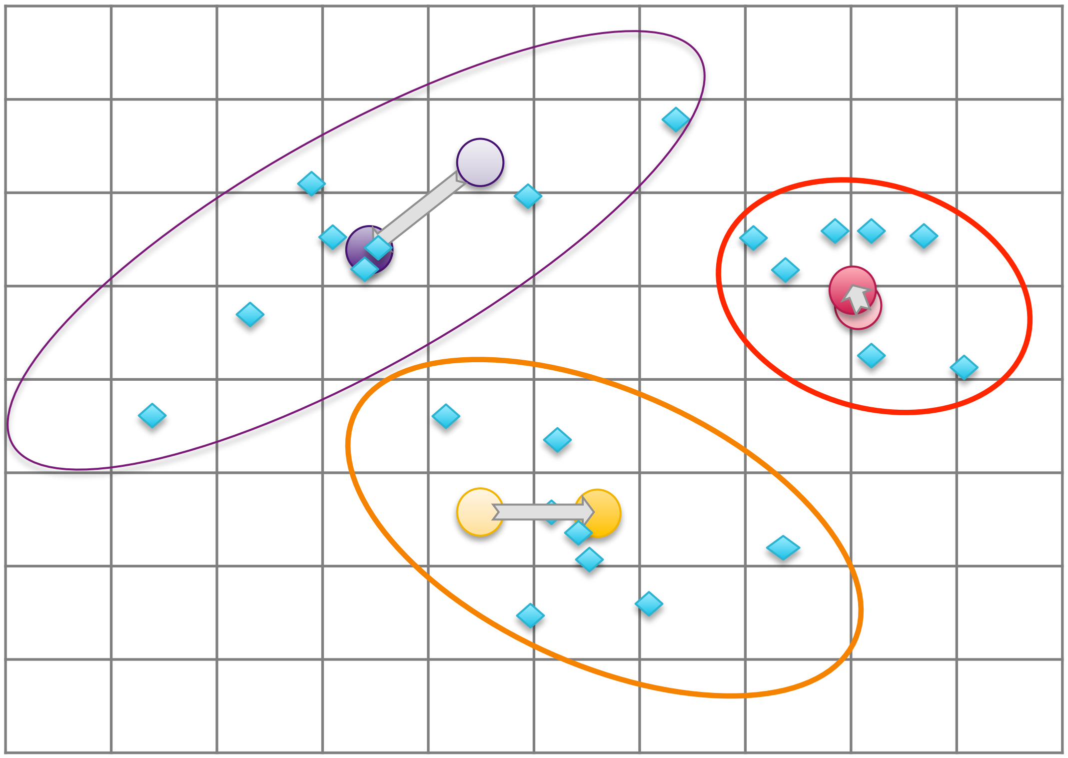
1. Choose *K* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed, iterate again

Example: k-means Clustering (7)



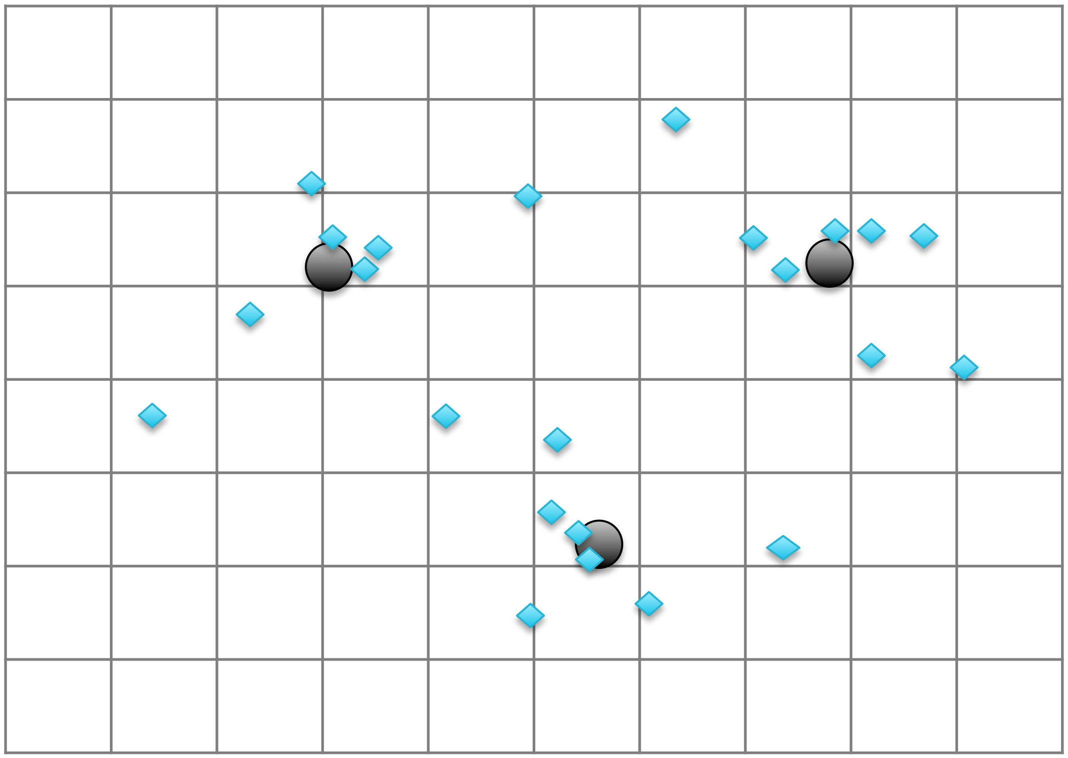
1. Choose *K* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed, iterate again

Example: k-means Clustering (8)



1. Choose *K* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed, iterate again

Example: k-means Clustering (9)

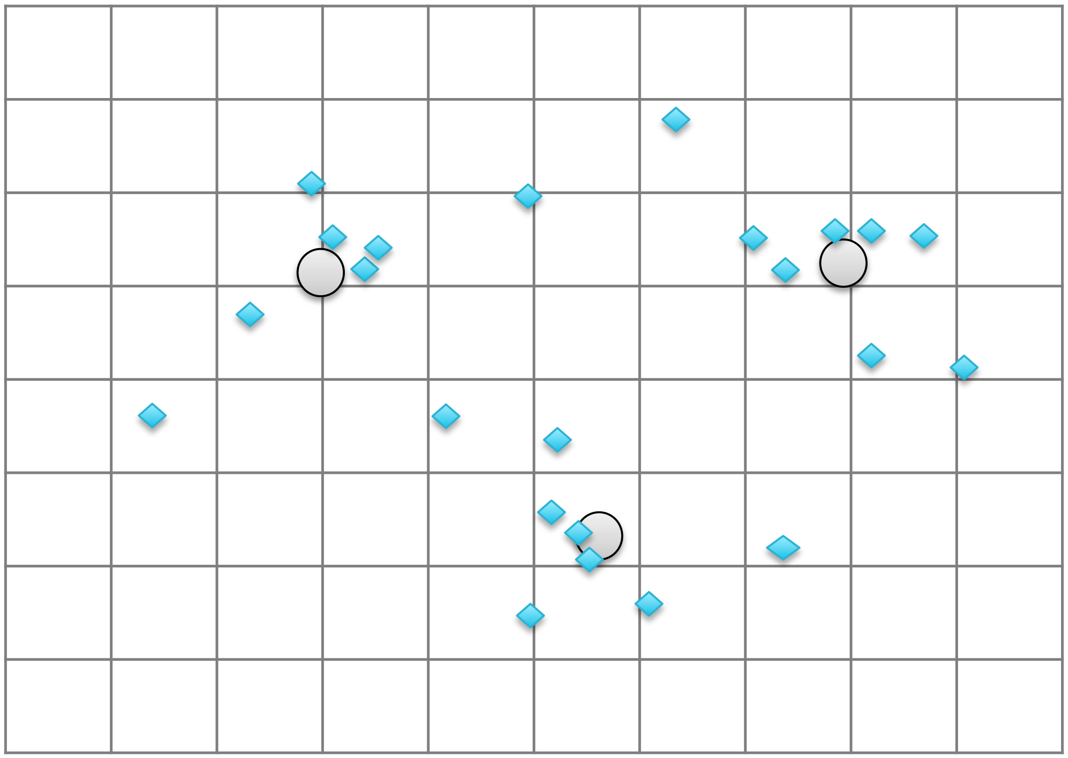


1. Choose *K* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed, iterate again

…

5. Done!

Example: Approximate k-means Clustering



1. Choose *K* random points as starting centers
2. Find all points closest to each center
3. Find the center (mean) of each cluster
4. If the centers changed by more than *c*, iterate again

…

5. Close enough!

Chapter Topics

Common Patterns in Spark Data Processing	**Distributed Data Processing with Spark**

- Common Spark Use Cases
- Iterative Algorithms in Spark
- Graph Processing and Analysis
- Machine Learning
- Example: k-means
- **Conclusion**
- Homework: Implement an Iterative Algorithm with Spark
- Optional Homework: Partition Data Files Using Spark

Essential Points

- **Spark is especially suited to big data problems that require iteration**
  - In-memory persistence makes this very efficient
- **Common in many types of analysis**
  - e.g., common algorithms such as PageRank and k-means
- **Spark includes specialized libraries to implement many common functions**
  - GraphX
  - MLlib
- **GraphX**
  - Highly efficient graph analysis (similar to Pregel et al.) and graph construction, representation and post-processing
- **MLlib**
  - Efficient, scalable functions for machine learning (e.g., logistic regression, k-means)

Chapter Topics

Common Patterns in Spark Data Processing	Distributed Data Processing with Spark

- Common Spark Use Cases
- Iterative Algorithms in Spark
- Graph Processing and Analysis
- Machine Learning
- Example: k-means
- Conclusion
- **Homework: Implement an Iterative Algorithm with Spark**
- **Optional Homework: Partition Data Files Using Spark**

Homework

- **Iterative Processing in Spark**
  - In this homework assignment you will
    - Implement k-means in Spark in order to identify clustered location data points from Loudacre device status logs
    - Find the geographic centers of device activity
- **Optional Homework: Partition Data Files Using Spark**
  - In this homework assignment you will
    - Define "regions" according to the k-means points identified above
    - Use Spark to create a dataset for device status data, partitioned by region
- **Please refer to the Homework description**