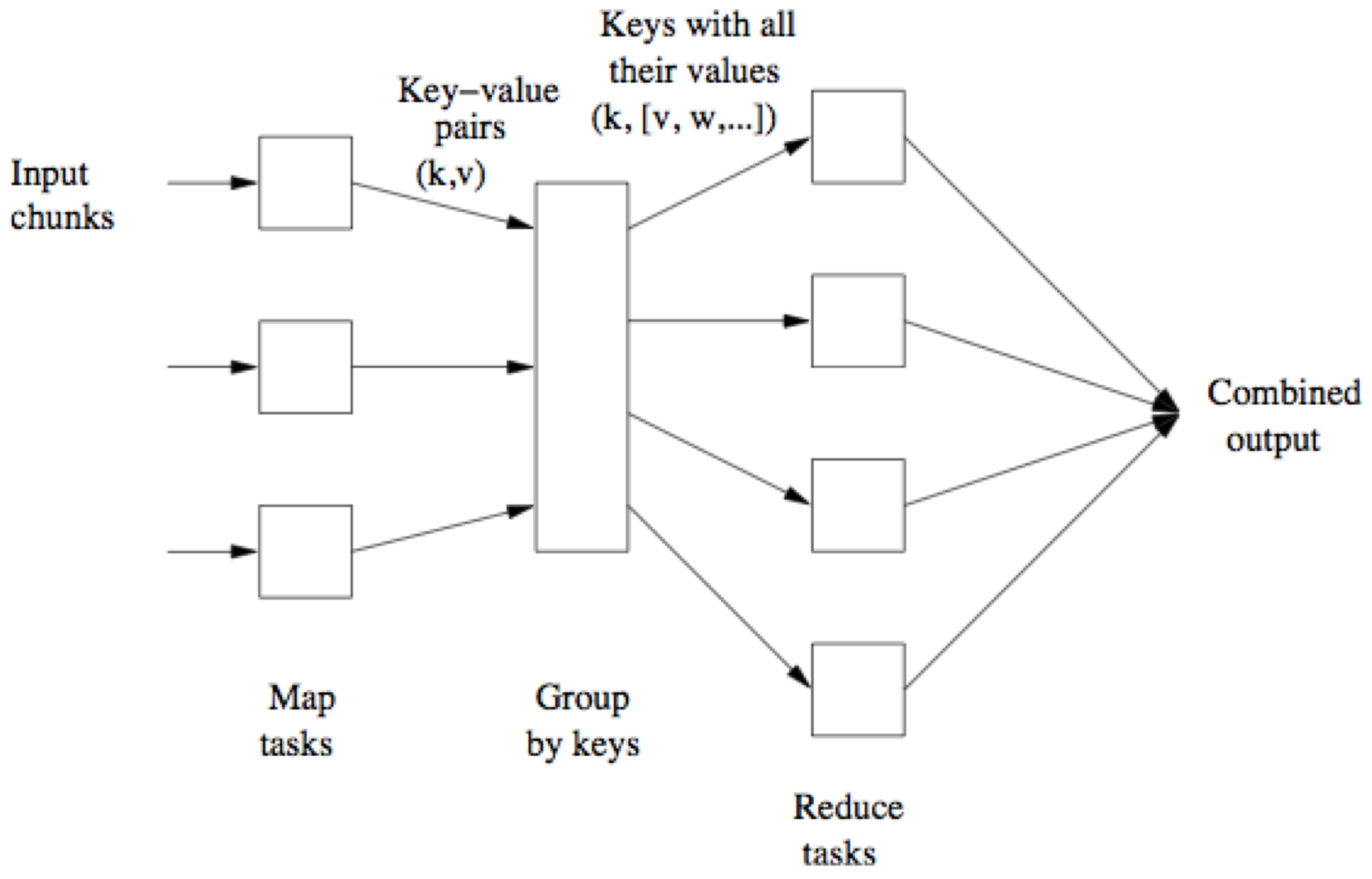
Introduction to MapReduce
(cont.)

Rafael Ferreira da Silva
rafsilva@isi.edu
http://rafaelsilva.com

USC Viterbi
School of Engineering
Information Sciences Institute

THE PROBLEM WITH AVERAGING STAR RATINGS

MapReduce: Summary



MapReduce: Examples

- Examples

```
$ docker run -it --rm -p 8888:8888 jupyter/pyspark-notebook
```

- Use file:

```
MapReduce-PySpark-Examples-2.ipynb
```

- Download **tip.json** from the yelp dataset challenge
  - https://www.yelp.com/dataset/challenge

Combiners

- Sometimes, a Reduce function is **associative** and **commutative**
  - Can be combined in any order, with the same result
  - Can push some of what Reducers do to Map task

- Reduces network traffic for data sent to Reduce task

- Example: word count
  - Instead of producing many pairs (w, 1), (w, 1), … can sum the n occurrences and emit (w, n)
  - Still required to do aggregation at the Reduce task for key, value pairs coming from multiple Map tasks

Example: Build an Inverted Index

Input:

tweet1, (“I love pancakes for breakfast”)
tweet2, (“I dislike pancakes”)
tweet3, (“What should I eat for breakfast?”)
tweet4, (“I love to eat”)

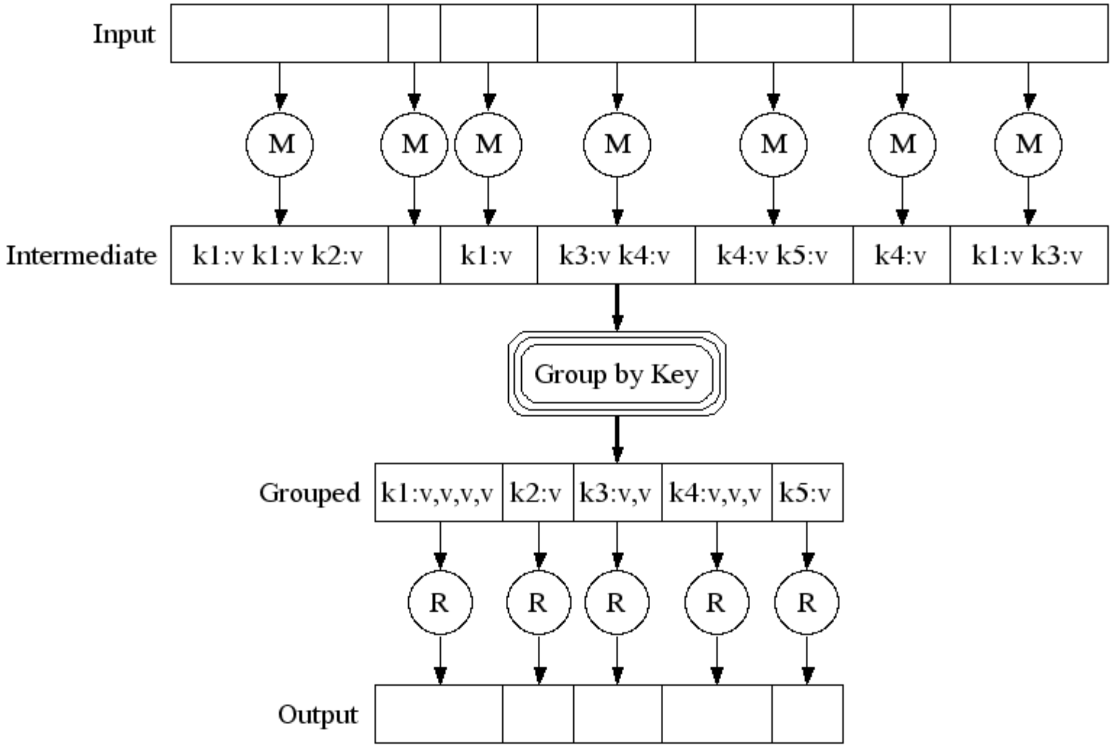
Desired output:

“pancakes”, (tweet1, tweet2)
“breakfast”, (tweet1, tweet3)
“eat”, (tweet3, tweet4)
“love”, (tweet1, tweet4)
…

Map task:
For each word, emit (word, tweetID) as intermediate key-value pair
Reduce task:
Reduce function then just emits key and list of tweetIDs associated with that key

MapReduce Environment

- MapReduce environment takes care of:
  - **Partitioning** the input data
  - **Scheduling** the program's execution across a set of machines
  - Performing the **group by key** step
    - In practice this is is the bottleneck
  - Handling machine **failures**
  - Managing required inter-machine **communication**



Data Flow

- **Input and final output** are stored on a **distributed file system (HDFS)**
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data

- **Intermediate results** are stored on **local FS** of Map and Reduce workers

- Output is often **input** to another MapReduce task

Coordination: Master

- **Master node takes care of coordination**
  - **Task status**: (idle, in-progress, completed)
  - **Idle tasks** get scheduled as workers become available
  - When a map task completes, it sends the **master** the location and sizes of its intermediate files
    - one for each reducer
  - Master pushes this info to reducers
- Master pings workers periodically to **detect failures**

Dealing with failures

- **Map worker failure**
  - Map tasks completed or in-progress at worker are <u>reset to idle and rescheduled</u>
  - Reduce workers are notified when task is rescheduled on another worker

- **Reduce worker failure**
  - Only in-progress tasks are reset to idle, why?
  - Reduce task is restarted

- **Master failure**
  - MapReduce task is aborted and client is notified

How many Map and Reduce jobs?

- ***M*** map tasks, ***R*** reduce tasks
- **Rule of a thumb**
  - Make ***M*** much larger than the number of nodes in the cluster
    - One DFS chunk per map task is common
  - Improves dynamic load balancing and speeds up recovery from worker failures
- **Usually *R* is smaller than *M***
  - Because output is spread across ***R*** files
- **Google example**
  - Often use 200,000 map tasks, 5000 reduce tasks on 2000 machines

Task granularity and pipelining

- **Fine granularity tasks:** map tasks >> machines
    - Minimizes time for fault recovery
    - Can do pipeline shuffling with map execution
        - Multiple mapper on same machine. One working, the other output/shuffling
    - Better dynamic load balancing

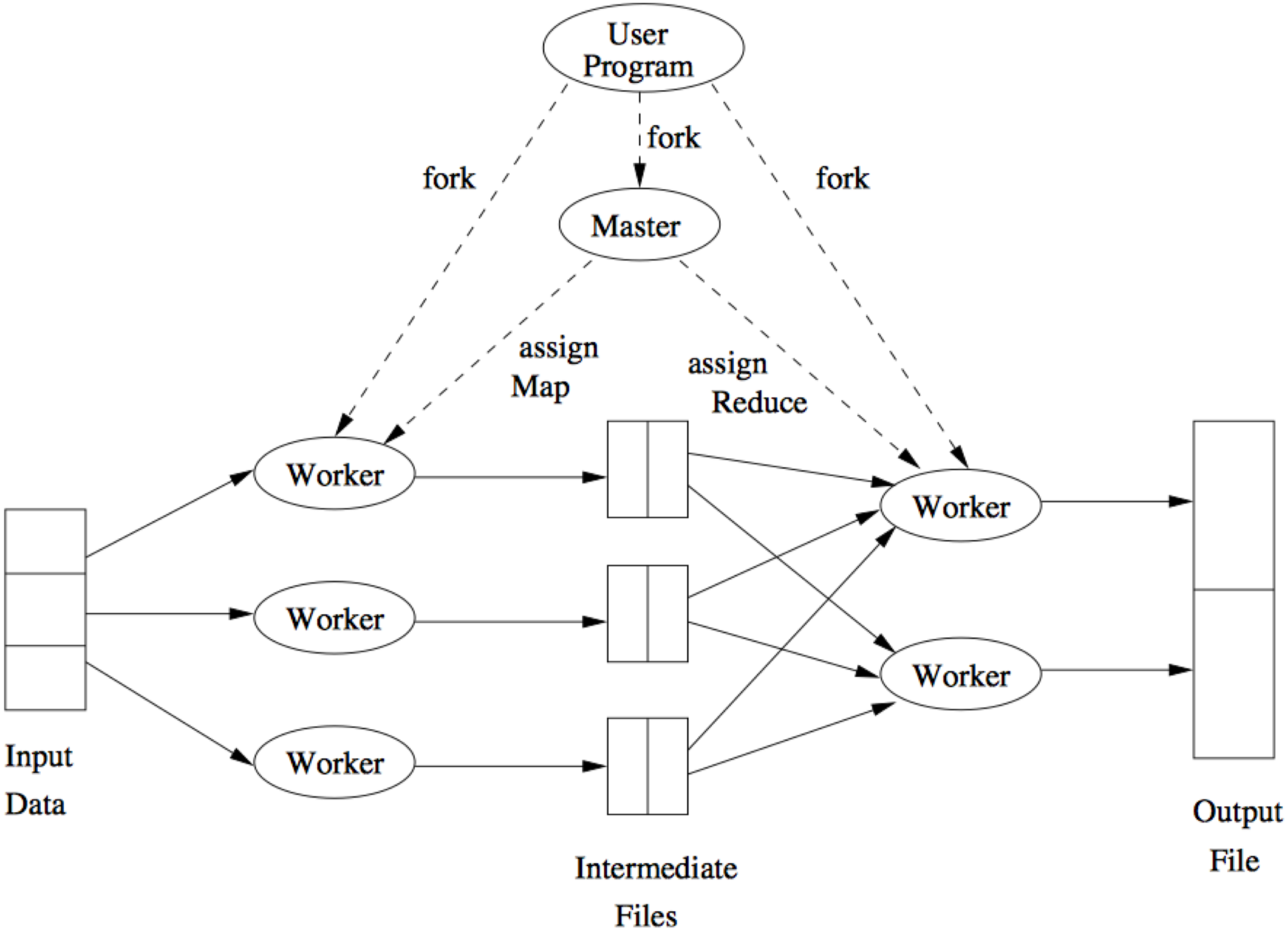


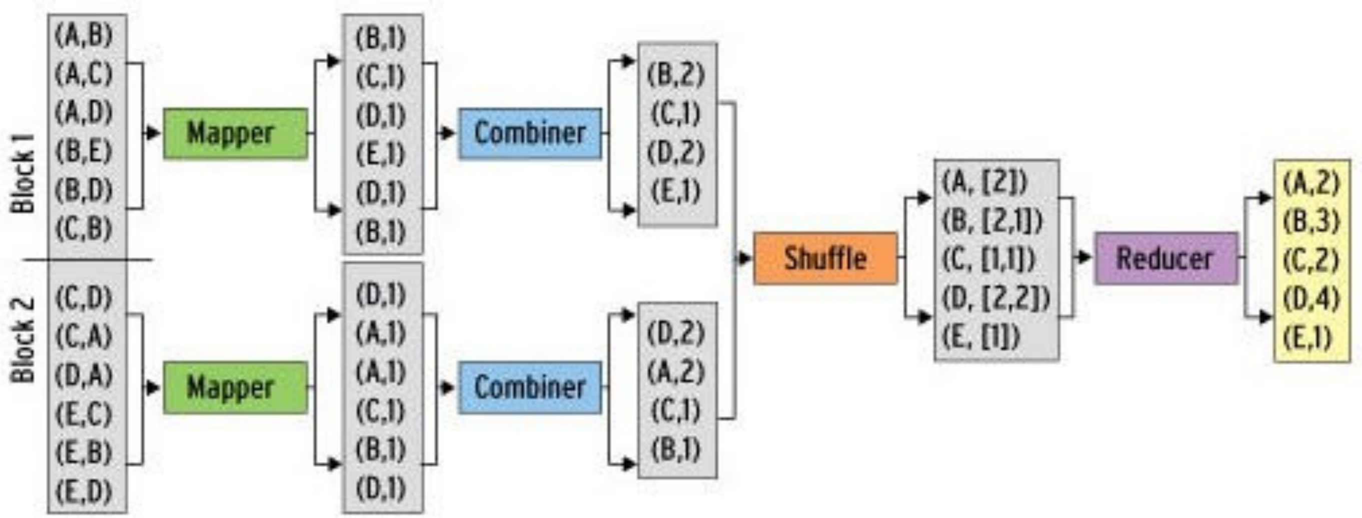
Figure 2.3: Overview of the execution of a MapReduce program

Refinements: Backup Tasks

- **Problem**
  - Slow workers significantly lengthen the job completion time
    - Other jobs on the machine
    - Bad disks
    - Weird things
- **Solution**
  - Near end of phase, spawn backup copies of tasks
    - Whichever one finishes first "wins"
- **Effect**
  - Dramatically shortens job completion time

Refinements: Combiners

- Back to our word counting example
  - **Combiner** combines the values of all keys of a single mapper …



  - Much less data needs to be copied and shuffled!
  - Works if reduce function is **commutative and associative**

Refinement: Partition function

- Want to control how keys get partitioned
  - Inputs to map tasks are created by **contiguous splits of input file**
  - Reduce needs to ensure that **records with the same intermediate key** end up at the same worker
- **System uses a default partition function**
  - **hash(key) mod *R***
- Sometimes useful to override the hash function
  - E.g., want to have alphabetical or numeric ranges going to different Reduce tasks (sorting)
  - E.g., **hash(hostname(URL)) mod *R*** ensures URLs from a host end up in the same output file

Problems Suited for MapReduce

General Characteristics of Good Problems for MapReduce

- Data set is truly “big”
  - **Terabytes**, not tens of gigabytes

  - Hadoop/MapReduce designed for terabyte/petabyte scale computation

  - Most real-world problems process less than 100 Gbytes of input
    - Microsoft, Yahoo: median job under 14 GB
    - Facebook: 90% of jobs under 100 GB

Source: “To Hadoop or Not to Hadoop” by Anand Krishnaswamy 8/13/2013

General Characteristics of Good Problems for MapReduce

- Don’t need **fast response time**

- When submitting jobs, Hadoop latency can be **a minute**

- Not well-suited for problems that require **faster response time**
  - online purchases, transaction processing

- A good **pre-computation engine**
  - E.g., pre-compute related items for every item in inventory

Source: “To Hadoop or Not to Hadoop” by Anand Krishnaswamy 8/13/2013

General Characteristics of Good Problems for MapReduce

- Good for applications that work in **batch mode**
- Jobs run without human interaction, intervention
- **Runs over entire data set**
  - Takes time to initiate, run; shuffle step can be time-consuming
- Does not support real time applications well: sensor data, real-time advertisements, etc.
- Does not provide good support for **random access to datasets**
  - Extensions: Hive, Dremel, Shark, Amplab

Source: "To Hadoop or Not to Hadoop" by Anand Krishnaswamy 8/13/2013

General Characteristics of Good Problems for MapReduce

- Best suited for data that can be expressed as **key-value pairs** without losing context, dependencies
- **Graph data harder to process using MapReduce**
  - Implicit relationships: edges, sub-trees, child/parent relationships, weights, etc.
- Graph algorithms may need information about the entire graph for each iteration
  - Hard to break into independent chunks for Map tasks
- Alternatives: Google's Pregel, Apache Giraph

Source: "To Hadoop or Not to Hadoop" by Anand Krishnaswamy 8/13/2013

General Characteristics of Good Problems for MapReduce

Other problems/data ***Not*** suited for MapReduce

- Tasks that need results of intermediate steps to compute results of current step
  - **Interdependencies among tasks**
  - Map tasks must be independent
- Some machine learning algorithms
  - Gradient-based learning, expectation maximization

Source: “To Hadoop or Not to Hadoop” by Anand Krishnaswamy 8/13/2013

General Characteristics of Good Problems for MapReduce

Summary: Good candidates for MapReduce

- Jobs that have to process **huge quantities of data** and either **summarize or transform the contents**

- Collected data has elements that can easily be captured with an identifier (key) and corresponding value

Source: "To Hadoop or Not to Hadoop" by Anand Krishnaswamy 8/13/2013

Useful tools for Data Science

Python libraries for data science

Library Name	Type
matplotlib	Visualization
Bokeh	Visualization
plotly	Visualization
Seaborn	Visualization
pydot	Visualization
scikit learn	Machine learning
XGBoost LightGBM CatBoost	Machine learning

Library Name	Type
eli5	Machine learning
SciPy	Data wrangling
NumPy	Data wrangling
pandas	Data wrangling
StatsModels Statistics in Python	Statistics
TensorFlow	Deep learning
PYTORCH	Deep learning

Library Name	Type
Keras	Deep learning
dist-keras elephas spark-deep-learning	Distributed deep learning
Natural Language ToolKit	NLP
spaCy	NLP
gensim	NLP
Scrapy	Data scraping

https://activewizards.com/blog/top-20-python-libraries-for-data-science-in-2018/

Pandas

https://pandas.pydata.org

- Python library that provides high-level data structures and a vast variety of tools for analysis

- Ability to translate rather **complex operations** with data into one or two commands

- Pandas contains many **built-in methods**
  - Grouping
  - Filtering
  - Combining data
  - Missing data
  - Time-series
  - Hierarchical indexing

Scikit-learn

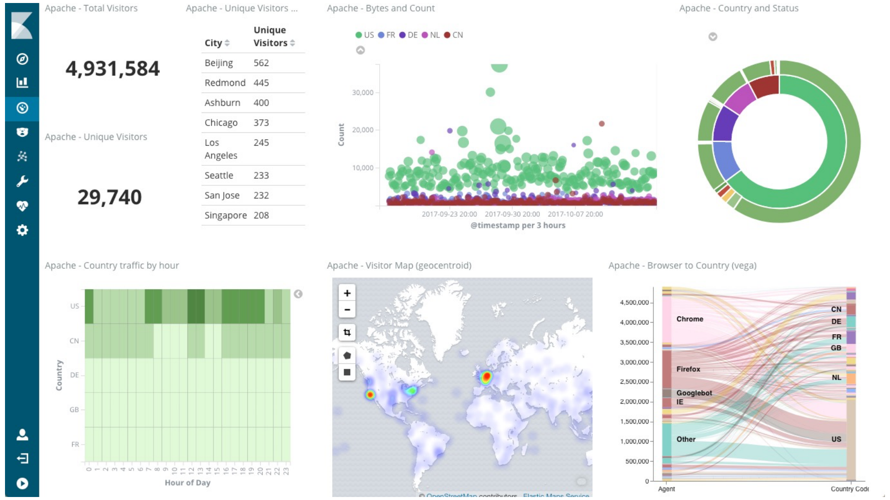
http://scikit-learn.org/stable/

- Python module based on **NumPy** and **SciPy**
- Provides algorithms for many standard **machine learning** and **data mining** tasks
  - Classification
  - Regression
  - Clustering
  - Dimensionality reduction
  - Model selection
  - Preprocessing

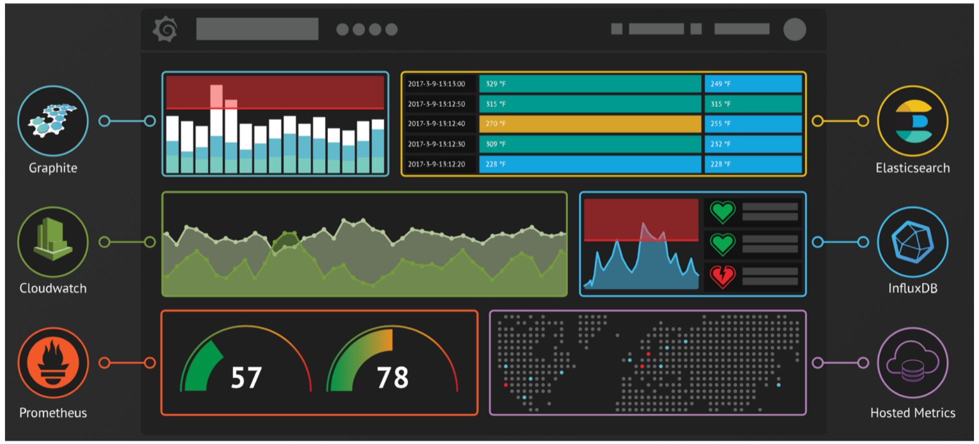
Elasticsearch

https://www.elastic.co/products/elasticsearch

- Open-source, RESTful, distributed search and analytics engine
  - **Real-time** data and real-time analytics
  - Scalable, high-availability, multi-tenant
  - **Full text** search



https://www.elastic.co/products/kibana



https://grafana.com

D3.js

- Data Driven Documents
  - JavaScript library for manipulating documents based on data

https://d3js.org