Muppet: MapReduce-Style processing for fast data

December 22, 2013

Outline

1. MapReduce framework
   - MapReduce Theory
   - MapReduce System
2. Muppet
   - Motivation
   - Muppet Framework

Outline

1. MapReduce framework
   - MapReduce Theory
   - MapReduce System
2. Muppet
   - Motivation
   - Muppet Framework

Function Objects

Definition

A function object is a function that can be manipulated as objects. e.g. Comparator objects used in c++ stl sort function.

An Example

```
struct myclass {
  bool operator() (int i,int j) {
      return (i<j);
   }
} myobject;
std::sort (myvector.begin(), myvector.end(), myobject);
```

Function Objects

Definition

A function object is a function that can be manipulated as objects. e.g. Comparator objects used in c++ stl sort function.

An Example

```
struct myclass {
   bool operator() (int i,int j) {
         return (i<j);
    }
} myobject;
std::sort (myvector.begin(), myvector.end(), myobject);
```

Function Objects

Definition

A function object is a function that can be manipulated as objects. e.g. Comparator objects used in c++ stl sort function.

An Example

```
struct myclass {
   bool operator() (int i,int j) {
         return (i<j);
    }
} myobject;
std::sort (myvector.begin(), myvector.end(), myobject);
```

Fold

Definition

Fold is a function that take a function object f and a list L as an input and recursively applies f to "combine" the elements of L
$fold(f, L[i:j]) = f(L[i], fold(f, L[i+1:j]))$

An Example

$fold(/)[64, 8, 4, 2] -> 64/(8/(4/2)) -> 16$

Fold

Definition

Fold is a function that take a function object f and a list L as an input and recursively applies f to “combine” the elements of L
$fold(f, L[i:j]) = f(L[i], fold(f, L[i+1:j]))$

An Example

$fold(/)[64, 8, 4, 2] -> 64/(8/(4/2)) -> 16$

Fold

Definition

Fold is a function that take a function object f and a list L as an input and recursively applies f to "combine" the elements of L
$fold(f, L[i:j]) = f(L[i], fold(f, L[i+1:j]))$

An Example

$fold(/)[64, 8, 4, 2] -> 64/(8/(4/2)) -> 16$

Map

Definition

Map is a function that take a function object f and a list L an input and applies f to each element of L to produce another list.
$map : f, L[i,j] -> [f(i), f(i+1), ..., f(j)]$

An Example

$map : sqrt, [1,4,9,16] -> [1,2,3,4]$

Map

Definition

Map is a function that take a function object f and a list L an input and applies f to each element of L to produce another list.
$map : f, L[i,j] -> [f(i), f(i+1), ..., f(j)]$

An Example

$map : sqrt, [1,4,9,16] -> [1,2,3,4]$

Map

Definition

Map is a function that take a function object f and a list L an input and applies f to each element of L to produce another list.
$map : f, L[i,j] -> [f(i), f(i+1), ..., f(j)]$

An Example

$map : sqrt, [1, 4, 9, 16] -> [1, 2, 3, 4]$

MapReduce

Definition

$mapreduce(f_m, f_r,) = reducePerKey(f_r, group(map(f_m, L)))$

$reducePerKey = fold(f_r, L_{key})$

MapReduce folds over a sorted result of a map

MapReduce

Definition

$mapreduce(f_m, f_r,) = reducePerKey(f_r, group(map(f_m, L)))$
$reducePerKey = fold(f_r, L_{key})$
MapReduce folds over a sorted result of a map

MapReduce System

1. Programming model to express computations such that the resulting program is "easily" parallelizable.
    1. The parallelization is taken care of by an algorithm rather than a programmer.

1. Associated system that allows executing programs based on the MR programming model on a cluster of commodity machines.
    1. Programmer only needs to write map and reduce functions and set few configuration parameters.
    2. The MapReduce fairy (system) takes care of everything else. (Hides the details of parallelization, failures, communication between processes etc.)

MapReduce System

1. Programming model to express computations such that the resulting program is “easily” parallelizable.
   1. The parallelization is taken care of by an algorithm rather than a programmer.

1. Associated system that allows executing programs based on the MR programming model on a cluster of commodity machines.
   1. Programmer only needs to write map and reduce functions and set few configuration parameters.
   2. The MapReduce fairy (system) takes care of everything else. (Hides the details of parallelization, failures, communication between processes etc.)

MapReduce System

1. Programming model to express computations such that the resulting program is "easily" parallelizable.
    1. The parallelization is taken care of by an algorithm rather than a programmer.

1. Associated system that allows executing programs based on the MR programming model on a cluster of commodity machines.
    1. Programmer only needs to write map and reduce functions and set few configuration parameters.
    2. The MapReduce fairy (system) takes care of everything else. (Hides the details of parallelization, failures, communication between processes etc.)

Outline

1. MapReduce framework
   - MapReduce Theory
   - MapReduce System

2. Muppet
   - Motivation
   - Muppet Framework

MapReduce System

1. Map: Takes input key value pair and outputs a set of "intermediate" key value pairs.
2. The MapReduce framework groups the intermediate key value pairs and produces key, value list.
3. Reduce: Takes key, value list and summarizes the list.

MapReduce System

1. Map: Takes input key value pair and outputs a set of "intermediate" key value pairs.
2. The MapReduce framework groups the intermediate key value pairs and produces key, value list.
3. Reduce: Takes key, value list and summarizes the list.

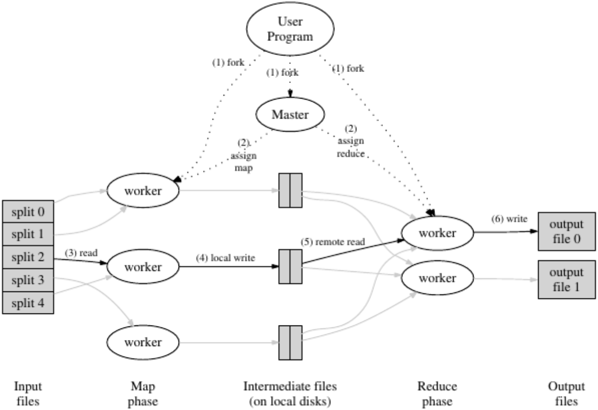
MapReduce System

1. Map: Takes input key value pair and outputs a set of "intermediate" key value pairs.
2. The MapReduce framework groups the intermediate key value pairs and produces key, value list.
3. Reduce: Takes key, value list and summarizes the list.

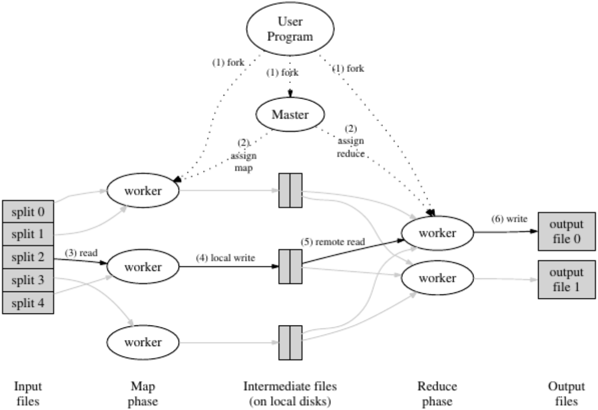
MapReduce System

1. Map: Takes input key value pair and outputs a set of "intermediate" key value pairs.
2. The MapReduce framework groups the intermediate key value pairs and produces key, value list.
3. Reduce: Takes key, value list and summarizes the list.

Execution Overview



Execution Overview



Outline

1. MapReduce framework
   - MapReduce Theory
   - MapReduce System
2. Muppet
   - Motivation
   - Muppet Framework

Streaming Data

1. The data continuously keeps flowing at high speed.
2. Challenge: Compute summary information based on the high speed streaming data

Examples

a) How to detect "hot topics" on twitter quickly, given the high speed stream of all public tweets.
b) Given a foursquare checkin stream, maintain the count of checkins per retailer.

Streaming Data

1. The data continuously keeps flowing at high speed.
2. Challenge: Compute summary information based on the high speed streaming data

Examples

a) How to detect "hot topics" on twitter quickly, given the high speed stream of all public tweets.
b) Given a foursquare checkin stream, maintain the count of checkins per retailer.

Streaming Data

1. The data continuously keeps flowing at high speed.
2. Challenge: Compute summary information based on the high speed streaming data

Examples

a) How to detect "hot topics" on twitter quickly, given the high speed stream of all public tweets.
b) Given a foursquare checkin stream, maintain the count of checkins per retailer.

Streaming Data

1. The data continuously keeps flowing at high speed.
2. Challenge: Compute summary information based on the high speed streaming data

Examples

a) How to detect "hot topics" on twitter quickly, given the high speed stream of all public tweets.
b) Given a foursquare checkin stream, maintain the count of checkins per retailer.

MapReduce deficiencies

1. MapReduce system needs to look at a snapshot of data
   1. Fresh data cannot be included in the middle of MapReduce execution
   2. There is no such thing as a snapshot in a stream
2. Given that the input stream is high speed, the recovery from failure should be very quick.

MapReduce deficiencies

1. MapReduce system needs to look at a snapshot of data
    1. Fresh data cannot be included in the middle of MapReduce execution
    2. There is no such thing as a snapshot in a stream
2. Given that the input stream is high speed, the recovery from failure should be very quick.

MapReduce deficiencies

1. MapReduce system needs to look at a snapshot of data
   1. Fresh data cannot be included in the middle of MapReduce execution
   2. There is no such thing as a snapshot in a stream
2. Given that the input stream is high speed, the recovery from failure should be very quick.

Outline

1. MapReduce framework
   - MapReduce Theory
   - MapReduce System
2. Muppet
   - Motivation
   - Muppet Framework

Objectives

1. Easy to program
2. Manage dynamic datastructures and make them quereable
3. High speed processing of streaming data
4. Should be easy to scale up by throwing machines at the growing data size

Objectives

1. Easy to program
2. Manage dynamic datastructures and make them quereable
3. High speed processing of streaming data
4. Should be easy to scale up by throwing machines at the growing data size

Objectives

1. Easy to program
2. Manage dynamic datastructures and make them quereable
3. High speed processing of streaming data
4. Should be easy to scale up by throwing machines at the growing data size

Objectives

1. Easy to program
2. Manage dynamic datastructures and make them quereable
3. High speed processing of streaming data
4. Should be easy to scale up by throwing machines at the growing data size

Objectives

1. Easy to program
2. Manage dynamic datastructures and make them quereable
3. High speed processing of streaming data
4. Should be easy to scale up by throwing machines at the growing data size

Events and Streams

Events and Streams

Event e is a tuple $< sid, ts, k, v >$

sid- Stream ID that the e belongs to

ts- Global time stamps, to allow well defined merging of multiple streams

k- key that need not be unique across events, used to group events

v- value field

Definition

A Stream is a sequence of events in the increasing order of time stamp *ts*

Events and Streams

Events and Streams

Event e is a tuple $< sid, ts, k, v >$
sid- Stream ID that the e belongs to
ts- Global time stamps, to allow well defined merging of multiple streams
k- key that need not be unique across events, used to group events
v- value field

Definition

A Stream is a sequence of events in the increasing order of time stamp *ts*

Events and Streams

Events and Streams

Event e is a tuple $< sid, ts, k, v >$

sid- Stream ID that the e belongs to

ts- Global time stamps, to allow well defined merging of multiple streams

k- key that need not be unique across events, used to group events

v- value field

Definition

A Stream is a sequence of events in the increasing order of time stamp *ts*

Map Function

1. A map function subscribes to one or more streams.
2. Events are fed to the map function in the increasing order of time stamps ts
3. A map function takes an event as an input and produces zero or more events $map(event) \rightarrow event*$ to various streams

Map Function

1. A map function subscribes to one or more streams.
2. Events are fed to the map function in the increasing order of time stamps ts
3. A map function takes an event as an input and produces zero or more events $map(event) \rightarrow event*$ to various streams

Map Function

1. A map function subscribes to one or more streams.
2. Events are fed to the map function in the increasing order of time stamps ts
3. A map function takes an event as an input and produces zero or more events $map(event) \rightarrow event*$ to various streams

Map Function

1. A map function subscribes to one or more streams.
2. Events are fed to the map function in the increasing order of time stamps ts
3. A map function takes an event as an input and produces zero or more events $map(event) \rightarrow event*$ to various streams

Update Function

1. Input characteristics same as map function
2. An Update function $U(e, S_{U,k})$ is also given a slate $S_{U,k}$ along with the event e having a key k
3. The slate $S_{U,k}$ is an in memory datastructure used to keep all the summary information about the events with key k seen by U
4. The pair $< U, k >$ uniquely identifies a slate.
   1. $S_{U_1,k}$ and $S_{U_2,k}$ are two different slates even though the key is the same.
5. Slate is "live" in memory datastructure that is continuously updated with the streaming data

Update Function

1. Input characteristics same as map function
2. An Update function $U(e, S_{U,k})$ is also given a slate $S_{U,k}$ along with the event e having a key k
3. The slate $S_{U,k}$ is an in memory datastructure used to keep all the summary information about the events with key k seen by U
4. The pair $< U, k >$ uniquely identifies a slate.
    1. $S_{U_1,k}$ and $S_{U_2,k}$ are two different slates even though the key is the same.
5. Slate is “live” in memory datastructure that is continuously updated with the streaming data

Update Function

1. Input characteristics same as map function
2. An Update function $U(e, S_{U,k})$ is also given a slate $S_{U,k}$ along with the event e having a key k
3. The slate $S_{U,k}$ is an in memory datastructure used to keep all the summary information about the events with key k seen by U
4. The pair $< U, k >$ uniquely identifies a slate.
    1. $S_{U_1,k}$ and $S_{U_2,k}$ are two different slates even though the key is the same.
5. Slate is "live" in memory datastructure that is continuously updated with the streaming data

Update Function

1. Input characteristics same as map function
2. An Update function $U(e, S_{U,k})$ is also given a slate $S_{U,k}$ along with the event e having a key k
3. The slate $S_{U,k}$ is an in memory datastructure used to keep all the summary information about the events with key k seen by U
4. The pair $< U, k >$ uniquely identifies a slate.
    1. $S_{U_1,k}$ and $S_{U_2,k}$ are two different slates even though the key is the same.
5. Slate is "live" in memory datastructure that is continuously updated with the streaming data

Update Function

1. Input characteristics same as map function
2. An Update function $U(e, S_{U,k})$ is also given a slate $S_{U,k}$ along with the event e having a key k
3. The slate $S_{U,k}$ is an in memory datastructure used to keep all the summary information about the events with key k seen by U
4. The pair $< U, k >$ uniquely identifies a slate.
    1. $S_{U_1,k}$ and $S_{U_2,k}$ are two different slates even though the key is the same.
5. Slate is "live" in memory datastructure that is continuously updated with the streaming data

Update Function

1. Input characteristics same as map function
2. An Update function $U(e, S_{U,k})$ is also given a slate $S_{U,k}$ along with the event e having a key k
3. The slate $S_{U,k}$ is an in memory datastructure used to keep all the summary information about the events with key k seen by U
4. The pair $< U, k >$ uniquely identifies a slate.
    1. $S_{U_1,k}$ and $S_{U_2,k}$ are two different slates even though the key is the same.
5. Slate is "live" in memory datastructure that is continuously updated with the streaming data

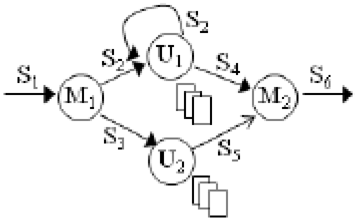
Overview

1. Memory less map function $map(event) \rightarrow event*$ and Update function with memory
2. Given an Update function slates partitioned based on event keys
3. Key space associated with the slate is partitioned by the number of machines running the update function.

Overview

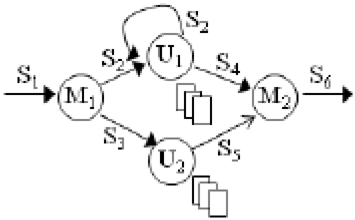
1. Memory less map function $map(event) \rightarrow event*$ and Update function with memory
2. Given an Update function slates partitioned based on event keys
3. Key space associated with the slate is partitioned by the number of machines running the update function.

MapUpdate Application



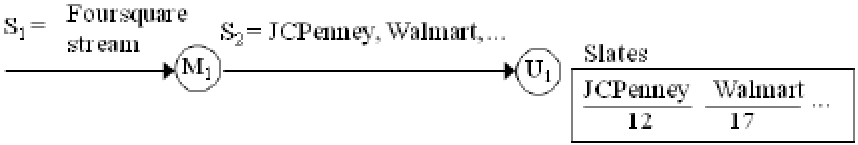
(a)

MapUpdate Application



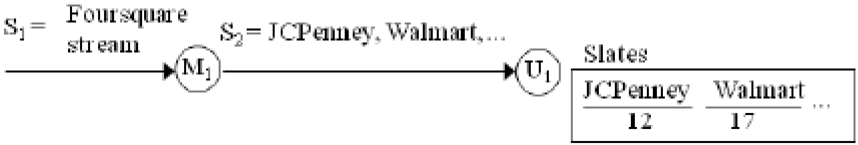
(a)

MapUpdate Application



(b)

MapUpdate Application



(b)