Naiad: A Timely Dataflow System

Derek G Murray et al.

24^{th} ACM Symposium on Operating System Principles, 2013

Presentation by:

SB Ramalingam Santhanakrishnan
K Kleeberger
B Jain

October 11, 2017 - Seminar Cloud Computing (IN4392)

Agenda

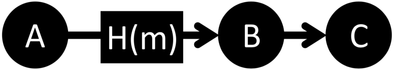
- Introduction
- Timely Data Flow

- Distributed Implementation
- Programming Model

- Performance Evaluation
- Real World Applications

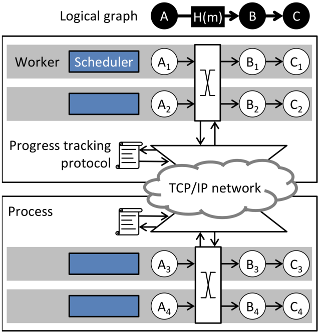
Distributed Implemenation

Naiad is the distributed implementation of timely dataflow



- A program consists of logical stages (A, B, C)
- H(m) controls exchange of data between stages

Data Parallelism



- Physical graph represents the chosen amount of workers and distributed connected hosts
- Programmer can select which way a message should flow in the system stages
- Naiad always uses the logical graph as a decision base where data has to flow

Worker

Delivers messages (data) and notifications to vertices

- Tie-breaker – Always deliver messages before notifications
- Responsible for multiple vertices
- Loop data does not pass trough the workers queue

Worker

Delivers messages (data) and notifications to vertices

- Tie-breaker – Always deliver messages before notifications
- Responsible for multiple vertices
- Loop data does not pass trough the workers queue

Synchronization

- Communicate through shared queue
- Queue not necessary if SEND/RECV are under same worker
- Re-entrancy due to loops – enqueue for later, coalesce incoming messages in OnRecv to reduce memory.

Progress tracking

Problem:

Notification can only be sent if there are no outstanding messages

Solution: Own progress tracking protocol

- Occurence Counters gets updated after a Broadcast to all workers
- Local counter never moves ahead of global counter
- Allows safe delivery of notifications

Optimizing broadcast updates

- Rely on the logical graph, not the physical.
- Buffer before broadcast.
- Optimistically first broadcast via UDP to reduce latency.
- Wake up threads with either broadcast or unicast with programming primitives.

Fault tolerance and availability

CHECKPOINT and RESTORE interface

- Vertex
  - either log data or
  - full checkpoint when requested
- Progress Tracking Protocoll
  - full checkpoint

Tradeoff between Performance and Durability

- paper favours performance over durablity
- Prelies on durable input and output

Micro Stragglers

Naiad is sensitive to latency but tiny interruptions can decrease overall performance

Iterative Computation	¡ 1 MS
GC, Package loss	10th of MS

Network

- Disable Nagle's algorithms
- Set smaller retry timeout for package loss
- Use different network protocols in datacenters for computing

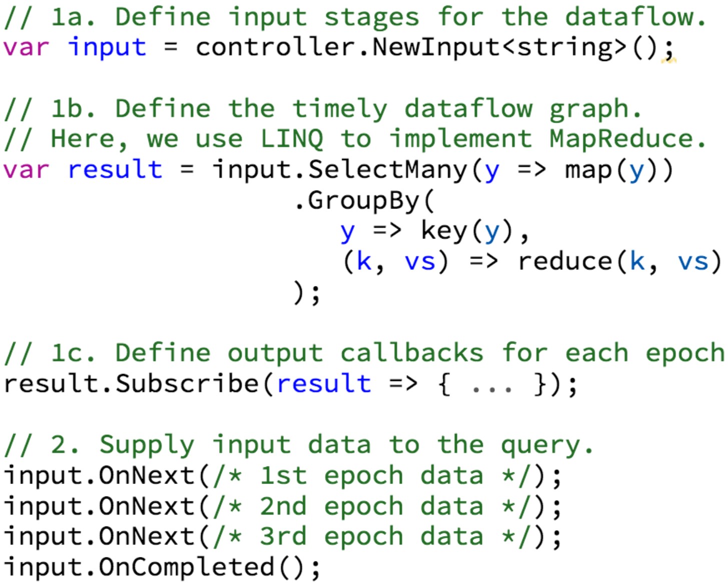
Garbage Collection

- avoid object allocation
- use buffer pools
- use value types

Naiad Program

- Provides public API with primitives
- Higher Level APIs
  - LINQ
  - MapReduce
  - Pregel
- Examples do not support coordination
  - to improve performance
  - concat, distinct, select
- Generic API for vertex programming
  - First define the behavior dataflow vertices
  - Second define the topology

Prototype Program



```
// 1a. Define input stages for the dataflow.
var input = controller.NewInput<string>();

// 1b. Define the timely dataflow graph.
// Here, we use LINQ to implement MapReduce.
var result = input.SelectMany(y => map(y))
                  .GroupBy(
                     y => key(y),
                     (k, vs) => reduce(k, vs)
                  );

// 1c. Define output callbacks for each epoch
result.Subscribe(result => { ... });

// 2. Supply input data to the query.
input.OnNext(/* 1st epoch data */);
input.OnNext(/* 2nd epoch data */);
input.OnNext(/* 3rd epoch data */);
input.OnCompleted();
```

Thank you

SB Ramalingam Santhanakrishnan

K Kleeberger

B Jain