A Scheduling Hub Service (SHS) for Application Data Transfers

draft-wang-alto-large-data-framework-01

Presenter: Haibin Song

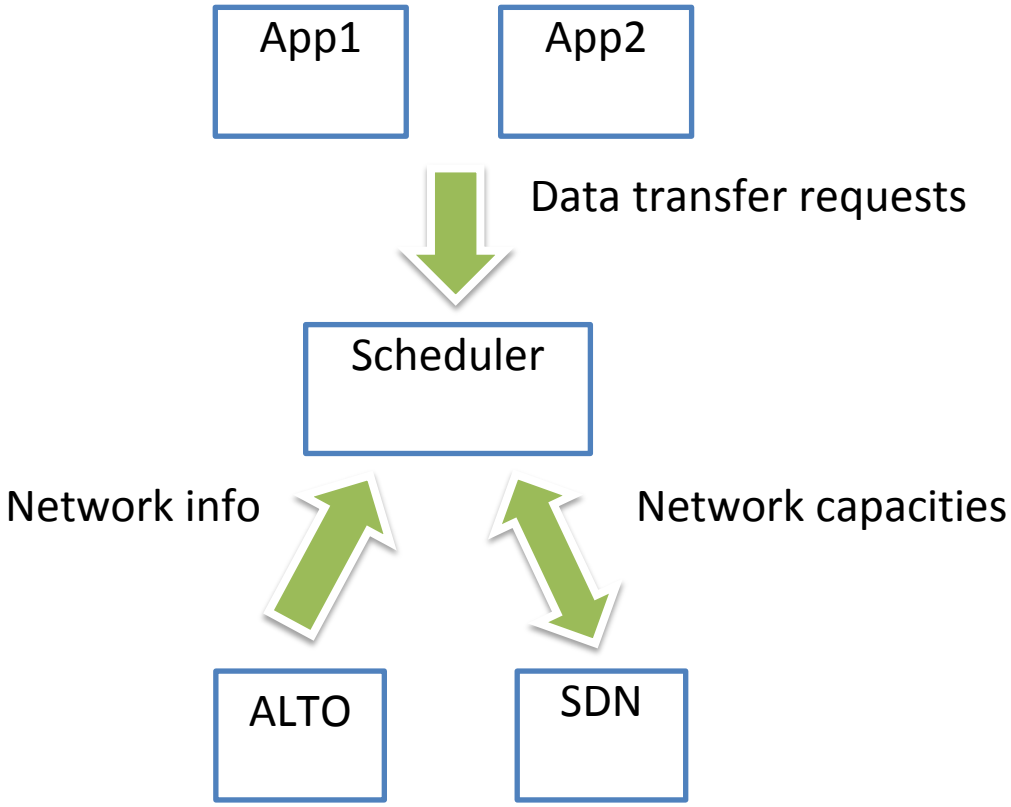
July 21, 2015 @ IETF 93

Problem

- A network may have a large number of large-data data transfer applications, e.g.,
  - Big data app (MapReduce, Spark)
  - Science data transfer
- Many duplicate functionalities, e.g.,
  - Application-layer traffic optimization (ALTO)
- Cross-app coordination is difficult

Proposed Solution

- A scheduling hub service to
  - implement common functionalities (reduce app complexity)
  - provide cross-app coordinations (achieve better network-wide utility)



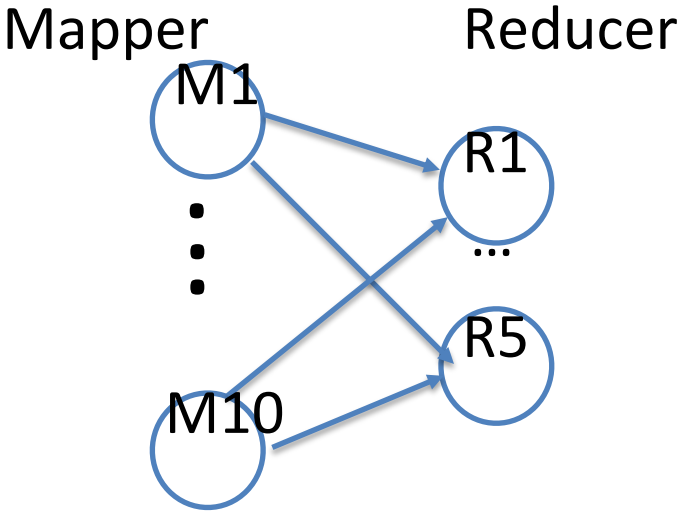
Key Design Points

- Service API
  - Simple, flexible, to capture application needs
- Scheduling algorithm
  - Able to collect and utilize network info (e.g., ALTO)
  - Able to utilize additional network capabilities (e.g., SDN customized routing)
- The focus of this document is the service API, which can benefit from standardization

Application Transfer is not As Simple As You Think

- Example: A MapReduce App

- Suppose an MR round has 10 mappers and 5 reducers. Each mapper transfers data to each reducer. There will be 50 transfers in all in the round.



MapReduce round goal: Minimize the finishing time of all transfers, not one individual transfer.

Service API: Requirements

- Allow application to dynamically manage data transfer jobs (e.g., add and remove jobs).
- Allow application to provide basic job information and requirements (e.g., file size, QoS timing).
- Allow application to convey dependency and coordination (e.g., MapReduce grouping).

Service API

- An application can create a set of jobs:
  - register() -> jobID: register a job
  - unregister(jobID): unregister a job

- Each job can contain a set of transfer tasks
  - createTaskDesc(type, [args]) -> task: create a task description
  - addTask(jobID, task) -> taskID: add a task to a job
  - removeTask(jobID, taskID): remove task by jobID and taskID

Basic Model

- Application Compute-Transfer Structure
  - Computation logic of application can be divided into several pieces of small (partial) data computations
  - Data computations are connected by data transfers

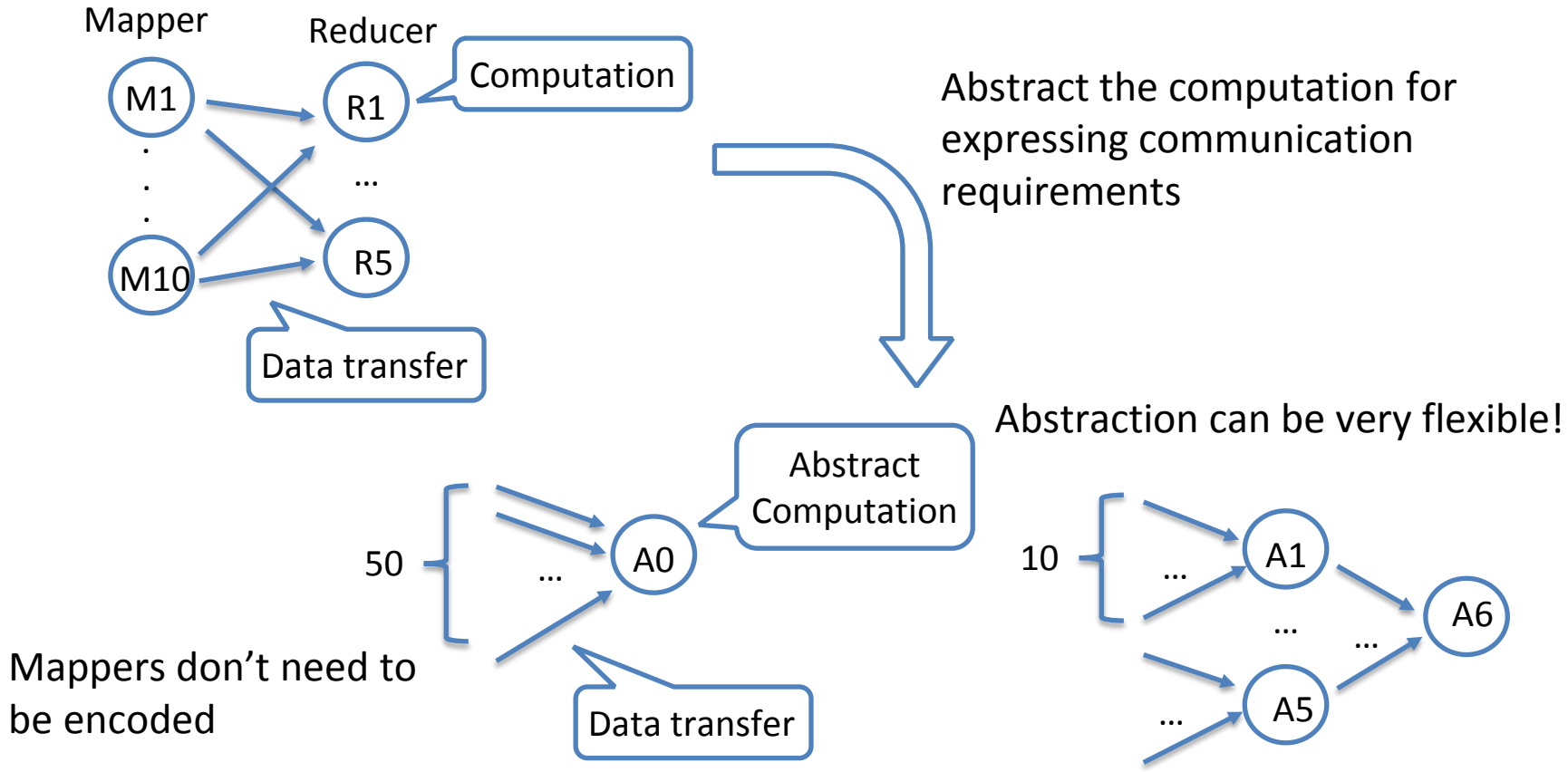
- Convey the structure to Directed Acyclic Graph (DAG) for expressing application requirements for SHS
  - Each node is a computation
  - Each link is a data transfer

For SHS, it does not need to know the exact computation.

- Abstract computation of nodes in DAG
  - Express the communication requirements:
    - Dependency type, e.g., all | one
    - Throughput matching
    - Pipelining, blocking
    - Deadline

Example: A MapReduce App

- Using Application Compute-Transfer Structure, a MapReduce job (the example before) can be expressed by 15 data computations and 50 data transfers, as shown below:



Map the Model to Design

- DataTransferTask (for link in DAG)
  - Manages the basic info for data transfer, like src, dst, file size, and offset
  - Should reflect the performance requirements by application, like deadline
- SyncTask (for node in DAG)
  - Set attributes for expressing the communication requirements

Task Details

- DataTransferTask:
  - src: the src of data transfer task
  - dst: the dst of data transfer task
  - dataSize: the size of data
  - offset: the offset of data
  - deadline: the deadline of the task

- SyncTask:
  - [dependencies]: a set of DataTransferTasks it depends on
  - [attributes]: the attributes of the task

API Example

Map Reduce:

```
val jobID = register ()

val task_1 = createTaskDesc("DataTransferTask", "src"="m1", "dst"="r1",
"dataSize"="100", "offset"="0")
…
val task_50 = createTaskDesc("DataTransferTask", "src"="m10", "dst"="r5",
"dataSize"="300", "offset"="0")
val task_0 = createTaskDesc("SyncTask", "dependences"=[task_1,…,task_50],
"dependency_type"="all")

val taskID_1 = addTask(jobID, task_1)
…
val taskID_50 = addTask(jobID, task_50)
val taskID_0 = addTask(jobID, task_0)
```

JSON: Map Reduce Example

```
{
      "job-id" : "00",
      "task": {
          "type" : "data-transfer-task",
           "src" : "http://192.168.0.0/bigdata/mapreduce/map0.data",
           "dst" : "http://192.168.1.0/bigdata/mapreduce/reduce0.data",
           "data-size" : "100", "offset" : "0"
         }
}

{
    "job-id" : "00",
    "task": {
        "type" : "sync-task",
          "dependencies" : [ "01", "02", … , "50"],
          "dependency_type" : "all"
     }
}
```

Application Scope of the Service

- Data Center
  - Big data applications (MapReduce, Spark) with customised requirements can optimise data transfer by the service.
- ISP
  - Non-real-time applications, such as backups, migration of data, can achieve efficient usage of bandwidth.

Backup Slides

Compared with Coflow, NetStitcher

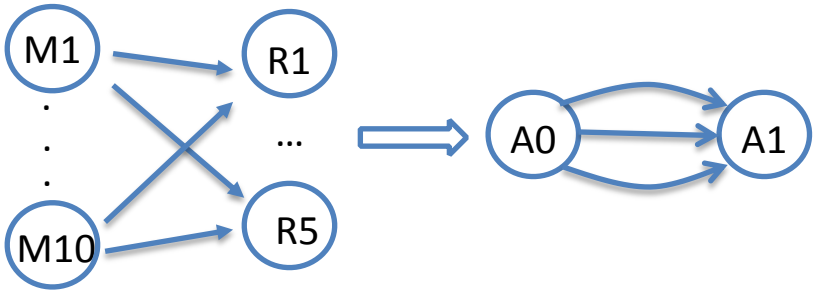
- SHS
  - DataTransferTask defines the basic info of data transfer
  - SnycTask defines the relations (e.g. dependency) between data transfers
- Coflow
  - Considers the flows in the coflow are independent
  - Dependency is between coflows

register(*numFlows*, [*options*]) ⟹ *coflowId*
put(*coflowId*, *dataId*, *content*, [*options*])
get(*coflowId*, *dataId*) ⟹ *content*
unregister(*coflowId*)

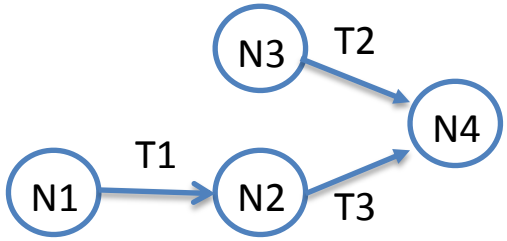
- NetStitcher
  - An overlay system comprising of a sender, intermediate nodes, and a receiver node
  - It schedules data transfers over the overlay

join(v)
leave(v)
send(v, u, F)

Compared with Coflow



The abstraction has the same effect as coflow



Coflow is difficult to set objective on N4 for coordinating T2 and T3, while it's easy for ours!

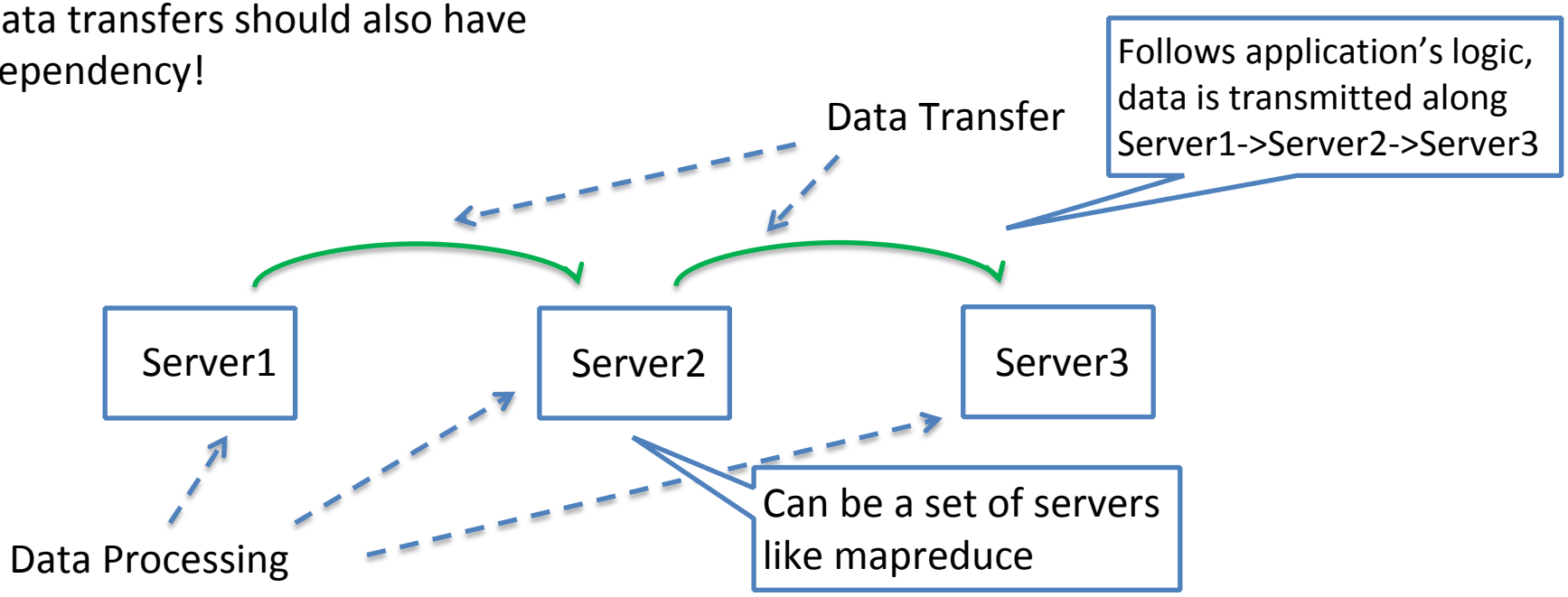
YANG Model

```
module: transfer-job
  +--rw job
     +--rw data-transfer-tasks* [task-id]
     |  +--rw task-id      task-id
     |  +--rw src?         uri
     |  +--rw dst?         uri
     |  +--rw dataSize?    int64
     |  +--rw offset?      Int64
     |  +--rw deadline?    time
     +--rw sync-tasks* [task-id]
     |  +--rw task-id      task-id
     |  +--rw dependencies*      task-id
     |  +--rw attributes* [attribute-type]
        |  +--rw attribute-type      string
        |  +--rw attribute-value     string
     +--rw job-id?              job-id
```

From Application's View

- Application focuses on the processing of the data
  - Data processing can be divided into several pieces based on the location of computing
  - Each piece should be linked by data transfer
  - Each piece depends on the former one

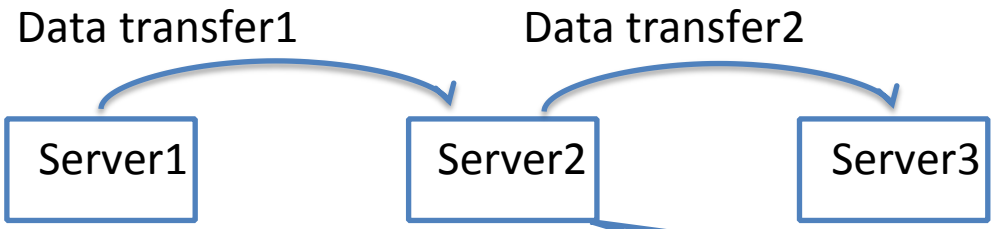
Data transfers should also have dependency!



Data Transfer Dependency

- The dependency of data transfer affects data processing at each server

- Each server along the path needs a synchronization to handle dependency for the correctness of data processing



All dependences need synchronization!

Needs a synchronization to ensure data transfer2 cannot start until dat a transfer1 finishes

The whole process can be denoted by several data transfers and several syn chronizations (connect data transfers)