ALTO WG
Internet-Draft
Intended status: Informational
Expires: January 21, 2016

X. Wang
S. Dong
Tongji University
G. Chen
Huawei Technologies

July 20, 2015

Design and Implementation of Large Data Transfer Coordinator
draft-wang-alto-large-data-framework-01.txt

Abstract

The Application-Layer Traffic Optimization (ALTO) protocol provides network information with the goal of improving both application performance and network resource utilization. As data transfers become larger (e.g., due to big data analysis), more data transfers are concurrent but with service requirements, and more network capabilities are emerging (e.g., SDN allowing a data transfer to request specific routes or Qos), the management of large data transfers has become an increasingly challenging issue. This document introduces Data Transfer Coordinator (DTC), a centralized data transfer scheduling framework which provides Scheduling Hub Service (SHS) to coordinate and schedule large data transfers. DTC considers all three components: data transfer requirements, (ALTO) network information, and SDN control capabilities. This document specifies not only the basic framework of DTC, but also a key component, service API for SHS to specify data transfers and their relations.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
2. Requirements Language 3
3. Terminology and Notation 3
4. Data Transfer Coordinator Framework 4
  4.1. Architecture 4
  4.2. Job Collector 5
  4.3. ALTO Client 5
    4.3.1. PASSIVE and ACTIVE Mode 6
  4.4. Task Scheduler 6
    4.4.1. Priority Model 6
  4.5. DTN Controller 7
5. Scheduling Hub Service 7
  5.1. Application Compute-Transfer Structure 8
  5.2. Abstract Computation 8
  5.3. DataTransferTask and SyncTask 9
  5.4. Service API 10
6. Example 11
7. Security Considerations 12
8. IANA Considerations 12
9. Acknowledgments 12
10. References 12
  10.1. Normative References 12
  10.2. Informative References 12
Authors' Addresses 13

1. Introduction

There is substantial need to manage large data transfers. Considering limited network resources such as bandwidth, inappropriate handling large data transfer would reduce performance significantly. It could be easier to cause network congestion than

low traffic. Congested network can result in higher rate of packet loss, then triggers retransmissions, which can cripple already heavily loaded networks. It's necessary to manage large data transfer not only for high network resource utilization but also for users' experience aspect.

Scheduling data flows needs network information such as available bandwidth between two transfer nodes. ALTO defines cost maps providing cost between two pids and endpoint cost service for two endpoints. By utilizing these network information, application can determine how to allocate bandwidth for each data flow. However, to archive such scheduling, there needs a centralized coordinator that can be aware of every data flow requirements. Moreover, to get the customized requirements for each data transfer, a general interface is need to obtain the correlation among data flows besides single data flow requirements.

This document introduces a centralized framework, Data Transfer Coordinator (DTC), which provides Scheduling Hub Service (SHS) for applications. SHS implements common functionalities for data transfers and provides cross-app coordination for achieving better network-wide utility. Also SHS provides a general API for applications to express data transfer relations by using two basic structures, DataTransferTask and SyncTask.

This document is organized as follows: Section 3 defines the Terminology and Notation in this document. Section 4 gives the details of SHS for scheduling large data transfer. Section 5 gives details of service API designed. Section 6 gives a MapReduce example for specifying relations between data transfers.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Terminology and Notation

This document uses the following additional terms:DTC, SHS, Job, Task.

o DTC

   Data Transfer Coordinator. A centralized framework includes Job Collector, Task Scheduler, ALTO Client, and DTN Controller to provide data transfer scheduling service to applications. See more detailed description in Section 4.

- SHS

  Scheduling Hub Service. Data transfer scheduling service considers both network information and data transfer requests. Data transfer requests are captured by two basic structures, DataTransferTask and SyncTask.See more detailed description in Section 5.

- Job

  Data transfer job that is registered by applications. A job includes tasks indicating data transfers and their relations submitted by one application. See more detailed description in Section 5.

- Task

  Including DataTransferTask and SyncTask that specifies data transfer information and their relations, respectively. See more detailed description in Section 5.

4. Data Transfer Coordinator Framework

4.1. Architecture

This section describes the design details of four components of the DTC framework, 1. Job Collector; 2. ALTO Client; 3. Task Scheduler; 4. Data Transfer Nodes (DTN) Controller. Among these four modules, task scheduler is the core of the framework. Job Collector provides interface to users for submitting data transfer requests, which will be passed to task scheduler for further process. Task scheduler makes scheduling based on the network information generated by ALTO client as well as the requirements of each data transfer from tasks. After computing allocation of bandwidth for each task, task scheduler will send transfer commands to DTN controller to start data transmission. Figure 1 shows the whole process.

```
                              .-----------.
                              |   Users   |
                              '-----------'
                                    | submit jobs
     .- - - - - - - - - - - - - - - | - - - - - - - - - - - - - - .
     |                        .-----------.                       |
     |                        |    Job    |                       |
     |         DTC            | Collector |                       |
     |                        '-----------'                       |
     |                             | pass user defined tasks      |
     |                             | to Task Scheduler            |
     | .-----------.          .-----------.          .---------.  |
     | |    DTN    |----------|   Task    |----------|   ALTO  |  |
     | | Controller|  send    | Scheduler |   get    |  Client |  |
     | '-----------' transfer '-----------' network  '---------'  |
     |               commands               state                 |
     ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -'
```

The benefits of DTC include:

o 1. It can achieve better network resource (bandwidth) allocation since it manages all data transfer requirements in a centralized framework.

o 2. It takes customized data transfer requirement into consideration by introducing DataTransferTask and SyncTask to capture correlation among data flows.

o 3. It's modular to support different scheduler algorithm implementations.

4.2. Job Collector

The job collector is responsible to manage data transfer requests from user and pass them to task scheduler for further process. It is important that the requests are dynamic and hence the API of the job collector allows dynamic insertion and deletion of data transfers. Details of the data transfer description and APIs for users are described in Section 5.3: Service API.

4.3. ALTO Client

ALTO client will be responsible to get network state to task scheduler for further usage. Although different scheduling algorithms may request different ALTO services, cost map and endpoint cost map seems to be the most useful services for scheduling tasks.

4.3.1. PASSIVE and ACTIVE Mode

ALTO client should support two modes according to the way it perceives network state changes, PASSIVE and ACTIVE. In PASSIVE mode, ALTO client will query ALTO server periodically to get latest network states. If the network state changes after one query, the ALTO client will not be aware of the change until next query. In ACTIVE mode, ALTO client will only query ALTO server once to get the initial network state. If network state changes after that, the ALTO client will be notified by ALTO server so it does not have to query ALTO server again. Note that ACTIVE mode will only be supported by ALTO server with ALTO SSE implemented.

4.4. Task Scheduler

The duty of task scheduler is to assign tasks from job collector to proper data transfer nodes (DTNs), splitting a file to several partial files to different DTNs if necessary, and notify the DTN controller to initiate the transfer. We will not discuss specific algorithm in this document but we assume algorithms used by scheduler should take network states provided by ALTO client into consideration. Different schedulers may obey different principles, some schedulers aims to maximize the number of finished tasks while some try to transfer as much data as possible.

4.4.1. Priority Model

In this section, we proposed a schedule model based on priority. In this model, every task will be set a predefined priority value, e.g. LOW, MEDIUM and HIGH, to indicate how important it is. The principle of this model is that tasks with higher priority have the privilege to occupy more resources such as available bandwidth. If the priority is not set, the task must be set a default one. Things become tricky when user does not specify priority but an expected finish time instead. However, in this model, it is easy to be solved by transforming expected finish time to priority by following steps:

o 01. Assign the lowest priority to the task and schedule the task.

o 02. Calculate the task's estimated finish time. If the estimated finish time is longer than user specified finish time, increase the task priority by one and reschedule the task, else the schedule procedure completes.

o 03. Keep doing step 2 until either the schedule procedure completes or the task is assigned as highest priority. If the task is still not able to be finished, we will keep it as highest priority and transfer as much data as possible.

The specific algorithm used to adjust the resources according to the priority is not described in this document.

4.5. DTN Controller

DTN controller is only responsbile for two following functions:

- o 01. Receive and process instructions from task scheduler, e.g. starting a new transfer, aborting a running transfer and adjusting transfer parameters such as transfer rate or number of connections.

- o 02. Monitor transfer status and update status changes to task scheduler. If a transfer failed or finished, it should notify task scheduler the details for further scheduling.

If we assume task scheduler is a manager, then DTN controller are workers who focusing on its own job without caring anything else. DTN controllers are not able to communicate with each other, which means it does not have a global view. Since the DTN controller has to utilize DTNs to transfer data, it should be deployed either in a server able to access DTNs or in the DTNs themselves.

5. Scheduling Hub Service

Introducing a systematic description of data transfer for SHS is challenging. Although it is easy to describe each individual data transfer, this simple description method is not sufficient for a centralized data transfer coordinator because it is not capable of representing relations, e.g. dependencies, between different data transfers. To solve this problem, this section first introduces the concept of Application Compute-Transfer Structure (ACTS) that captures the computation logic of application. ACTS includes the two basic components, data computation and data transfer. We find that for many data processing applications, they are composed of several data computations and several data transfers by which data computations are linked as a complete data processing. For example, MapReduce job includes mappers and reducers as data computation components, and data transfers act as connections between mappers and reducers.

However, for SHS, it doesn't need the exact computation at data computation nodes, but the enough knowledge to reflect the dependency between data transfers. Hence, we provide the ability of abstracting computation to applications for expressing dependency anf coordination between data transfers. By abstracting data computation, application can define the relation between data

transfers to/from one data computation node or a cluster of nodes, for expressing coarse grained dependency.

Finally, to map the concept to the design, SHS service API includes two transfer task types, DataTransferTask and SyncTask, which defines the basic data transfer information and relations between data transfers, respectively.

5.1. Application Compute-Transfer Structure

For many applications, the whole data processing would be divided into several pieces of small data computations depending on the different roles of servers, e.g., the MapReduce job is divided into two types of tasks, mapper and reducer, based on the role of servers. All partial data computations are linked by data transfers which transmit the result of computation from one place to another. By the joint collaboration of all small data computations, the application achieves the specific data processing. Then we use Application Compute-Transfer Structure (ACTS) which includes data computation and data transfer to convey the computation pattern of application. The mapping from computation logic of application to ACTS should be very obvius since it only includes data computation and data transfer.

By using ACTS, the computation logic of application can be defined as several data computations and several data transfers which link data computations, i.e., a Directed Acyclic Graph (DAG), in which each node is data computation and each link is data transfer.

5.2. Abstract Computation

For SHS, it doesn't need to know the exact computation of each data computation nodes in ACTS. But to schedule data transfers submitted by different applications, SHS needs the information about the relation between data transfers, such as dependency and coordiantion. The relation between data transfers is defined at data computation nodes. To achieve a collaboration of multiple data computation, each data computation must rely on the result of others. The dependency of data computations defines the relation of data transfers which is needed by SHS. Hence, to express the relation of data transfers, for a better scheduling, application should abstract its data computations

In this document, we define some attributes (dependency type, throughput matching, pipelining or blocking, and deadline) that can be used for abstract computation. Dependency type includes two values, all and one, to specify when to start the output data trnsfer at data computation nodes. All indicates the output data transfer cannot start until all input data transfers (at the same data

computation node) finishes, and one indicates if one input data transfer finishes, it can start output data transfer instead of waiting for other input data transfers. Throughput matching will defines the throughout relation between input data transfers and output data transfers. E.g., application needs a higher throughput for output data transfers than input ones. Pipelining and blocking indicates whether should the output data transfers wait the finishing of input data transfers or not. Deadline specifies the deadline for add dependent data transfers.

5.3. DataTransferTask and SyncTask

In this section, we define two types of task for mapping the concept to design of service API. DataTransferTask defines the basic information of data transfers while SyncTask defines the relation between data tansfers, i.e., abstract computation.

The schema for DataTransferTask (dtt) representation is described as following:

```
object {
    ResourcePath src;
    ResourcePath dst;
    JSONNumber dataSize;
    JSONNumber offset;
    [JSONString deadline;]
} DataTransferTask;

object {
    JSONString dependencies<1..*>;
    Attributes attributes<1..*>;
} SyncTask;

object {
    JSONString ss_id;
    JSONString path;
} ResourcePath

object {
    JSONString -> JSONString;
} Attributes;
```

with fields:

o src

This field specifies the source of data transfer.

- dst

  This field specifies the destination of data transfer.

- ResourcePath

  This field identifies a unique resource in multiple storege systems. Since a storage system could be connected by multiple data transfer nodes, it is not accurate to identify a resource by server host and file path anymore. To solve this problem, DTC will assign every connected storage system a unique id. Thus, users can combine ss_id, which is the unique storage system id, and file_path, which indicates location of the file in the corresponding storage system, to identify a unique resource.

- dataSize

  This field specifies the size of data to transport.

- offset

  This field specifies the offset of data. This provides the flexibility to application to split the data and transport them separately.

- dependencies

  This field specifies the dependencies of the SyncTask. Mapping to the ACTS, dependency of a SyncTask is the input data transfer of a data computation node.

- attributes

  This field specifies the attributes of the SyncTask. Attributes is key-value that key is the attributes name and value is the attributes value. Attributes can be dependency type of throughput matching as described.

5.4. Service API

Normally, users will register transfer jobs to include all conrresponding DataTransferTasks and SyncTasks. While a transfer jon is running, the user should be able to add tasks to or remove tasks from the job dynamically. To enable these features, a job collector should provide the following five functions for user:

- register()

  This function creates a new transfer job. It must return a job id for user to identify the job created. If the creation fails, it must throw an error.

- unregister(job_id)

  This function aborts a running transfer job. It accepts a job_id parameter and must abort all tasks belonging to the job. The function return value should indicate if the abort action succeeds or not. If the job does not exist, it must throw an error.

- createTaskDesc(type, [args])

  This function creates a task description satisfying the structure defined above. Type argument specifies the type of task, DataTransferTask or SyncTask. Args list specifies the content of the task, for DataTransferTask, it includes src, dst, dataSize, offset, and deadline; for SyncTask, it includes dependencies and attributes. This function returns the specified task for further operations.

- addTask(job_id, task)

  This function adds a new task to a existing job. This function accepts a job_id and a task as parameters. It must return a task id for user to identify the added task. If the creation fails, it must throw an error.

- removeTaskS(job_id, task_id,)

  This function removes a task from a existing job. This function accepts a job_id and a task_id. The job_id and task_id will identify a unique task to be removed. The function return value should indicate if the remove action succeeds or not.

6. Example

Suppose a MapReduce job has 10 mappers and 5 reducers. Each mapper transfers data to each reducer. There will be 50 data transfers in all. Application wants to express its requirements that minimize the finishing time of all transfers, not one individual transfer. Here we give a JSON example to show what should be sent to job collector for adding a DataTransferTask and a SyncTask to existing transfer job. After application added a DataTransferTask to transfer job, it will receive a task_id to identify the task (task_01, ..., task_50). Then it will use those task_id to add a SyncTask.

```
   {
       "job-id": "job_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": "job_00",
       "task": {
           "type": "sync-task",
           "dependencies": [ "task_01", "task_02",..., "task_50" ],
           "dependency_type": "all"
       }
   }
```

7. Security Considerations

This document has not conducted its security analysis.

8. IANA Considerations

This document does not specified its IANA considerations, yet.

9. Acknowledgments

The authors thank discussions with Yicheng Qian.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
           Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/
           RFC2119, March 1997,
           <http://www.rfc-editor.org/info/rfc2119>.

10.2. Informative References

[RFC7285] Alimi, R., Ed., Penno, R., Ed., Yang, Y., Ed., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, DOI 10.17487/RFC7285, September 2014, <http://www.rfc-editor.org/info/rfc7285>.

Authors' Addresses

Xin Wang
Tongji University
4800 Cao'an Road, Jiading District
Shanghai
China

Email: xinwang2014@hotmail.com

Shu Dong
Tongji University
4800 Cao'an Road, Jiading District
Shanghai
China

Email: dongs2011@gmail.com

Guohai Chen
Huawei Technologies
101 Software Avenue, Yuhua District
Nanjing
China

Email: chenguohai@huawei.com