Geo-distributed Datacenter Filesystem

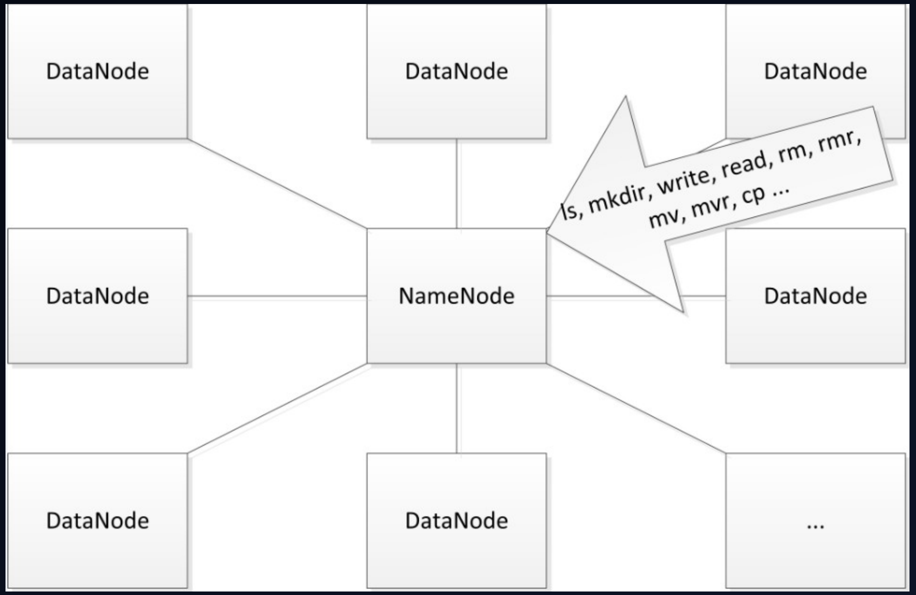
Overview

- Motivation
  - limitations of single index server
  - limitations of consistent hash
  - no geo-distributed datacenter filesystem
- Design
  - filesystem within only one datacenter
  - filesystem across multi datacenters
- Implementation
  - what I want to test
  - how to test

Motivation - limitations of single index server

GFS and HDFS keep a single index server separately to maintain the filesystem which incurs problems:

- single point failure
- performance bottleneck

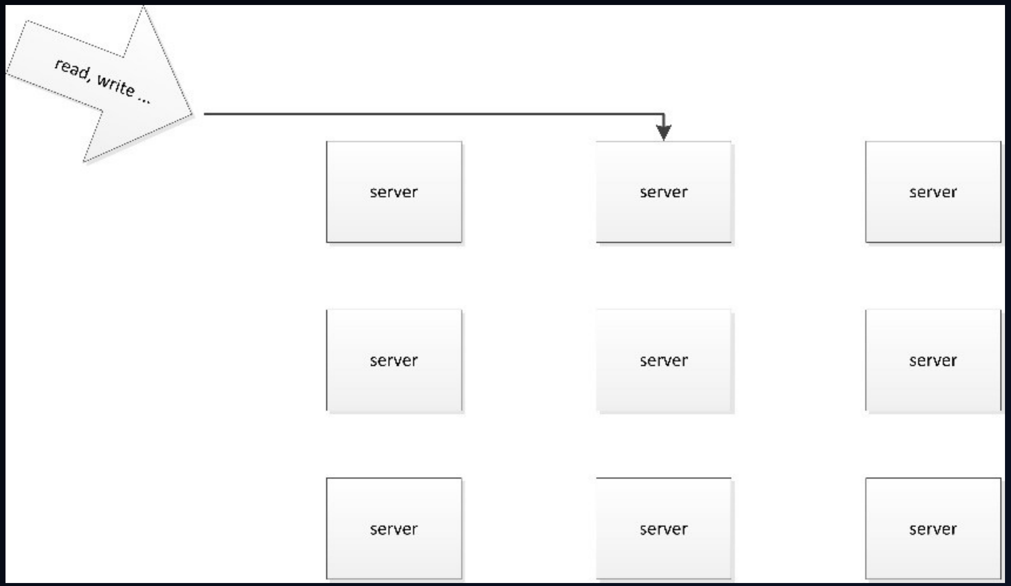


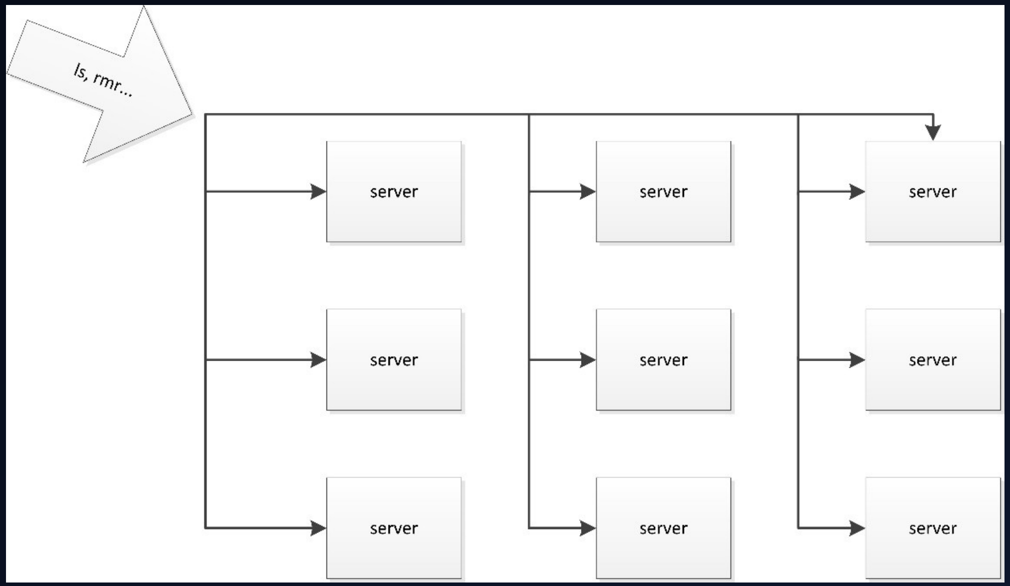
NameNode: “too many operations, I am overwhelmed”

Motivation - limitations of consistent hash

Consistent hash is unfit for maintaining the hierarchical filesystem:

- operations like *read, write, rm, cp, mv, mkdir, touch* could be done in O(1) while
- operations like *ls, rmr, mvr suffer*





Motivation – no geo-distributed datacenter filesy stem exists

If it exists, what would happen:

- common users are capable of moving about the country, getting access to their files without experiencing obvious latency.
- data analysis benefits from it due to data replication across data center, which shrinks the use of WAN for data transmission.
- multi-cloud storage and remote disaster-tolerant become more easy.

Design

- key idea:
  - separate operations into two parts
  - get access to all related files’ and directories' common parent directory in O(1)
  - make use of Othello to accelerate
- divided into two steps:
  - filesystem within one datacenter
  - filesystem across multi datacenters

Design - within one datacenter

- directories are stored as special files – directory file
- centralized panel is responsible for *mkdir,rmr,mvr* – these operations need to modify intermediate nodes from a filesystem tree view
- gateways are responsible for *ls,read,write,cp,mv,rm,touch* – these operation needn’t modify intermediate nodes or just need to modify leaf nodes from a filesystem tree view
- Othello is widely used in centralized panel, gateways and servers

Design - within one datacenter

Example1: mkdir path1

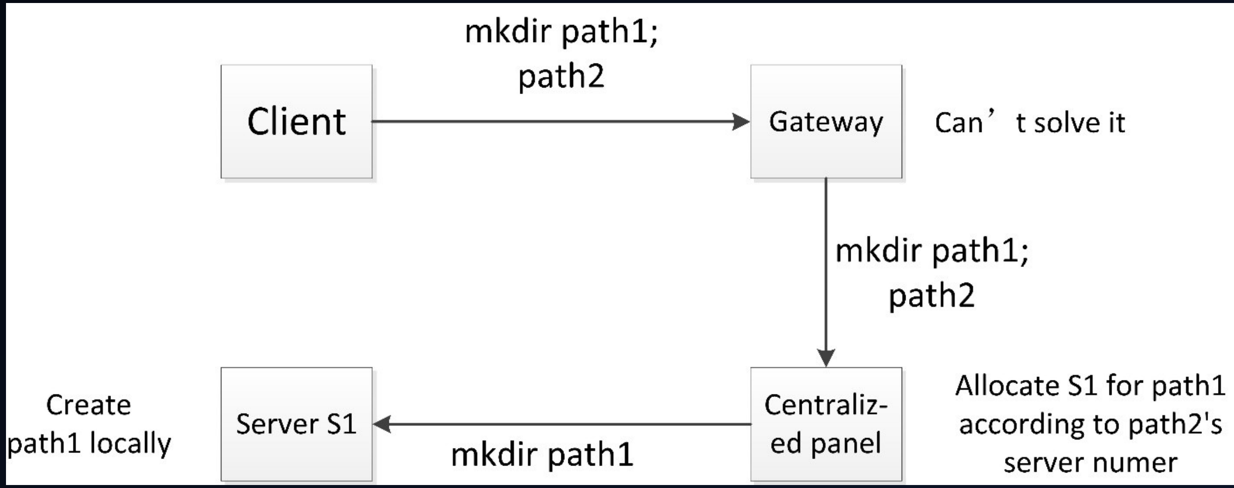
1. client tells gateway that *path1*'s parent directory is *path2*

2. gateway couldn't solve it and give it to centralized panel

3. centralized panel allocate one server S1 to *path1* to store path1's directory file according to path2's server number

4. S1 create directory path1 locally.

5. synchronize Othello in centralized panel to Othello in all gateways when appropriate



allocation algorithm in centralized panel would be clarified later

Design - within one datacenter

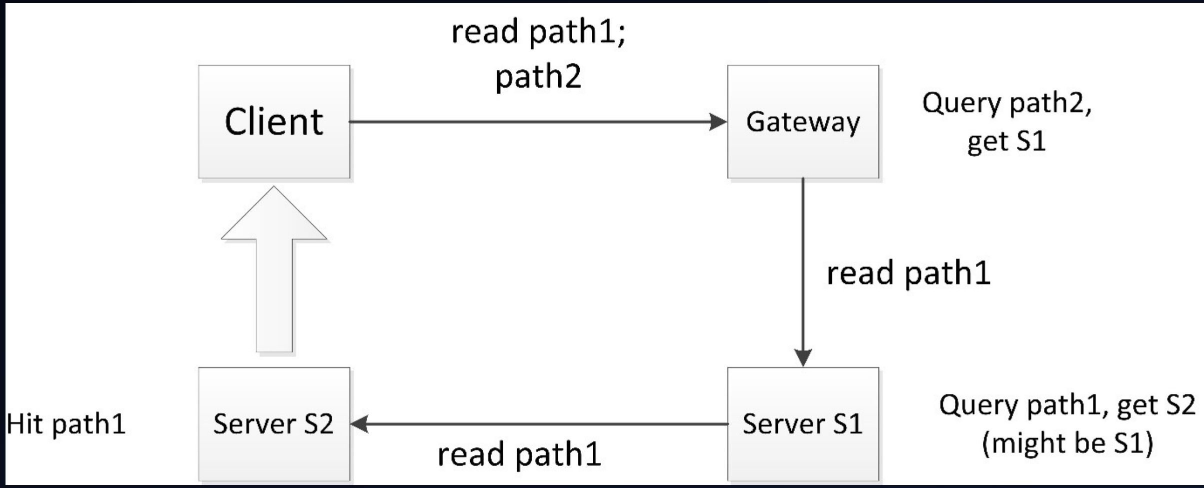
Example2: read path1

1. client tells gateway that *path1*'s parent directory is *path2*

2. gateway could solve it by firstly finding *path2*'s server S1 according to its Othello.

3. S1 continues to solve it by finding *path 1*'s server S2 (S2 might be S1)according to its Othello.

4. S2 responds with *path1*



Design - within one datacenter

From the view of Othello

- Othello (Control Structure) in centralized panel stores “directory path – server number”
- Othello (Query Structure)in gateway synchronizes with Othello in centralized panel periodically
- Othello in server stores “file path – server number”, in which files are direct children of directory stored in this server.

Design - within one datacenter

From the view of Othello

- Othello (Control Structure) in centralized panel stores “directory path – server number”
- Othello (Query Structure)in gateway synchronizes with Othello in centralized panel periodically
- Othello in server stores “file path – server number”, in which files are direct children of directory stored in this server.

Design - within one datacenter

Allocation algorithm in centralized panel

- assume p1's parent path is p2
- if p1 and p2 are stored in the same server, it does good for directory operations like *rmr.* But if this principle is the only one, all directory files would be stored in one server, which incurs single point failure and performance degradation.
- if p1 and p2 are always stored in different servers, directory operations like *rmr* have to visit tons of servers recursively.
- tradeoff

Design - within one datacenter

centralized panel在mkdir的时候能够获得的数据是：

1. 需要新建的目录的深度d
2. 需要新建目录的父目录所在的server#1
3. 数据中心中所有的server的情况，包括剩余存储空间A，文件与目录的数量比 F/D

逐一分析这三个因素，有的是阻止子目录与父目录在一起的，有的是欢迎子目录与父目录在一起的，具体为

1. d越大， 在一起，因为是纵向分割文件树
2. 剩余存储空间S越大，在一起
3. F/D越大，在一起，F/D越小，表明目录相比更多，将来更有可能有新的文件加入其中。引起load 的失衡

建立以下的线性模型决定子目录和父目录是不是分开, 假设父目录所在server编号为 i

dfile = d/ max(d)

A_i = S_i / Average(S)

R_i = (F_i/D_i) / Average(F/D)

T = alpha1 * dfile + alpha2 * A_i + alpha3 * R_i

当T低于阈值theta的时候，表明父目录所在server并不"欢迎"子目录，应该另找server

建立以下线性模型选择哪一个server存放子目录，对数据中心中的每一个server进行打分

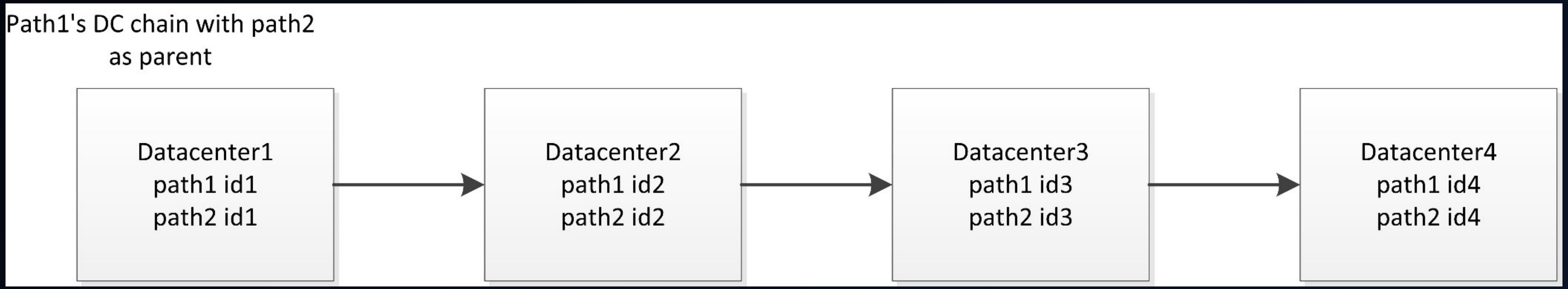
G = alpha4 * A_i + alpha5 * R_i + alpha6 * other factor

G越高得分越高，优先选择，其中other factor可以引入其他的因素，比如网络拓扑。虽然我们的假设是忽视数据中心内的延迟。但是如果能引入拓扑因素，比如树状拓扑，选择兄弟节点，Dcell拓扑，选择在同一个cell里的节点，或许可以取得更好的效果，当然也有可能变差(主要是load balance变差)

introduce data analyzers to customize to give better support for them

Design – across multi datacenters

- DC chain: a FIPO (enqueue in order, dequeue in random) r ecording datacenters (DCs) that store one directory or file and its replications (client).

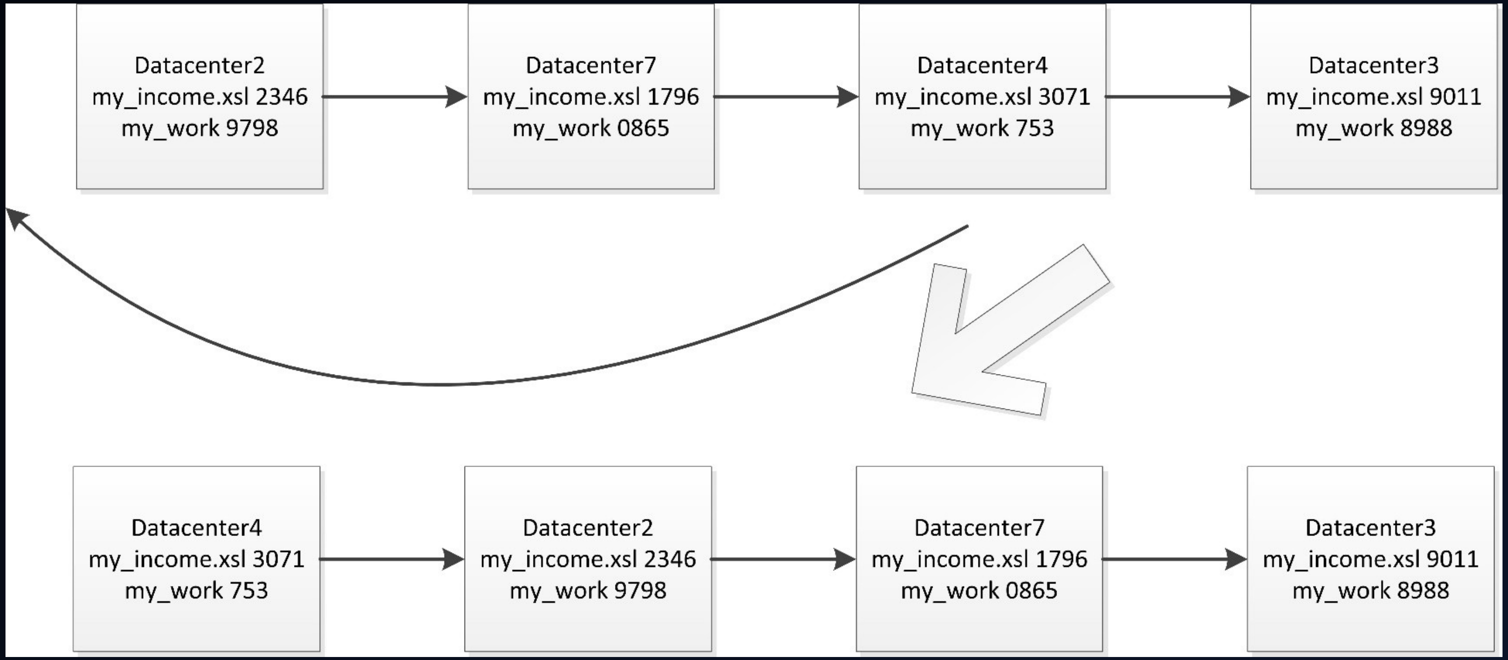


Design – across multi datacenters

- Othello in the centralized panel in different datacenters should be modified as:
- path - 0|xxxx : in this datacenter server xxxx
- path - 1|xxxs : in the datacenter xxxx and more information could be found from the Othello in the centralized panel of xxxx

Design – across multi datacenters

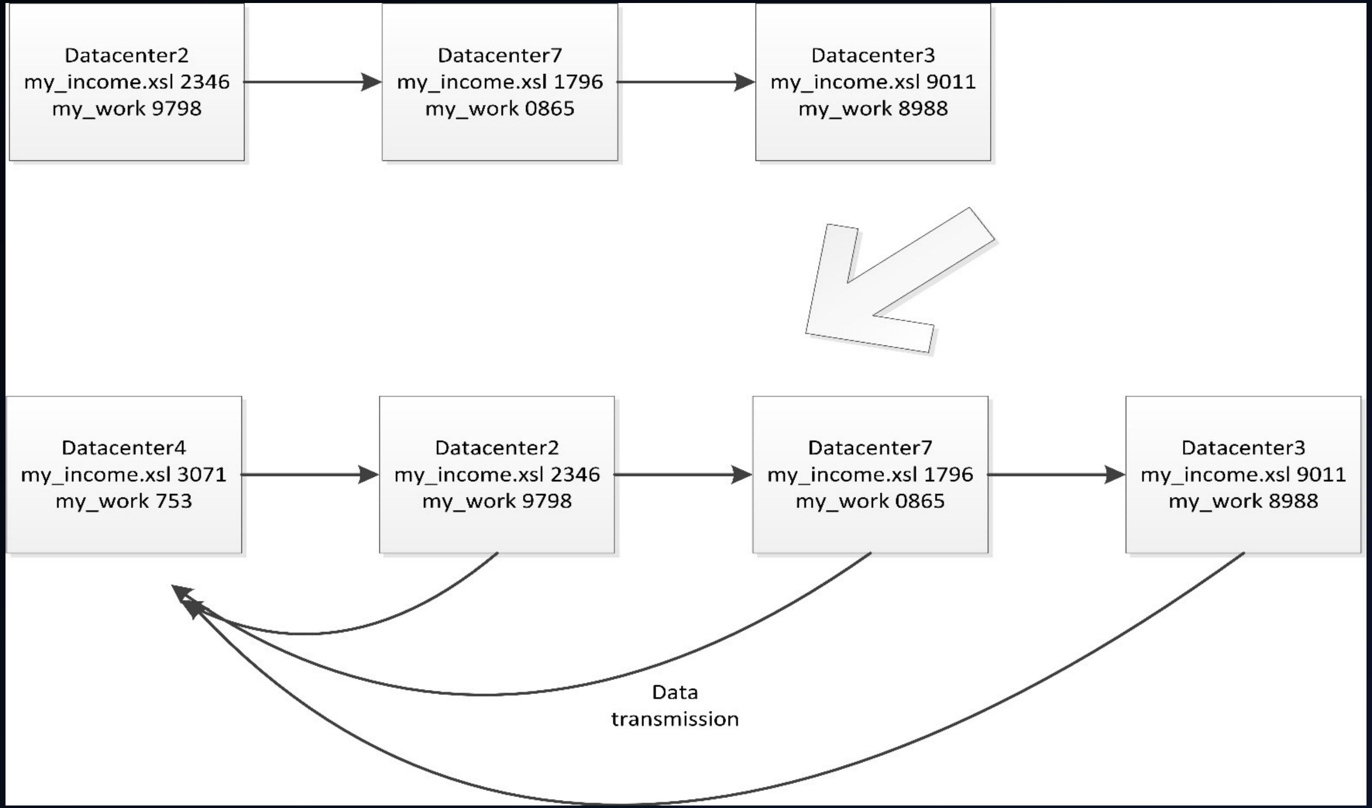
- *example1*: common user1 move from 成都 (served by datacenter2) to 上海 (served by datacenter4), the DC chain of his file my_income.xsl is directory my_job is as followed, and Othello in datacenter 2, 7 and 4 shall be changed



No data transmission !

Design – across multi datacenters

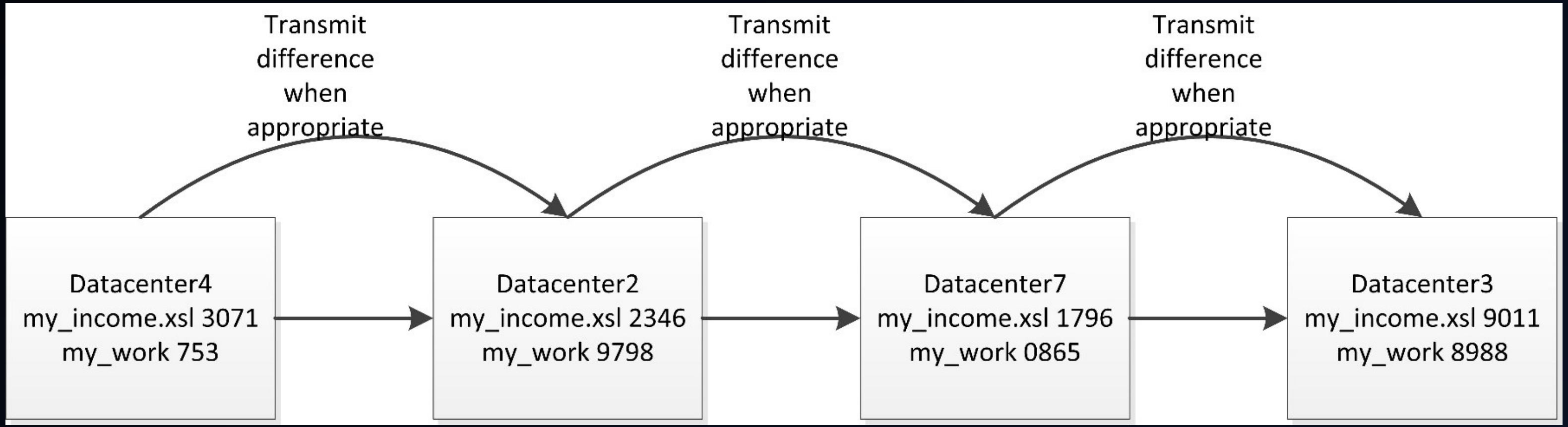
- *example2*: if common user1 move from 成都 (served by datacenter 2) to 上海 (served by datacenter4), and he/she have been to 上海 before



data transmission parallelly and data in DC3 could be deleted after it to save capacity

Design – across multi datacenters

- *example3*: write to my_income.xsl due to tripled salar
y!



update files in different datacenters step by step

Design – across multi datacenters

how to be compatible to current datacenters?

- place it at the tail of DC chain at the beginning
- pull data from it to new datacenters gradually when data is needed

Design – across multi datacenters

why my across mulit datacenters design works

- benefit common users
- benefit data analyzers – avoid abrupt data transmission through WAN by dispersing it to “appropriate time”
- benefit multi cloud storage – you needn’t revent wheel
- a practice of remote disaster-tolerance

Implementation – what I want to test

- using consistent hash as baseline
- draw comparison between the performance of *read, write, touch, rm, rmr, mv, mvr, mkdir, cp* in CH and these in my design in both one datacenter sc enario and multi datacenters scenario respectively
- test if files' depths would influence performance in my design
- draw comparison between fault tolerance and scalability in CH and these i n my design both one datacenter scenario and multi datacenters scenario respectively
- test compatibility of my design, that is to combine my design and CH to m ake up geo-distributed datacenters cluster and test possible performance degradation

Implementation – how to test

- from generating logs to a nalyzing logs
- from one datacenter sce nario to multi datacenter s scenario