LBVS:A Load Balancing Strategy for Virtual Storage

Hao Liu, Shijun Liu, Xiangxu Meng, Chengwei Yang, Yong Zhang
School of Computer Science & Technology
Shandong University
Shandong, Jinan, 250101

2010 International Conference on Service Sciences © IEEE

Outline

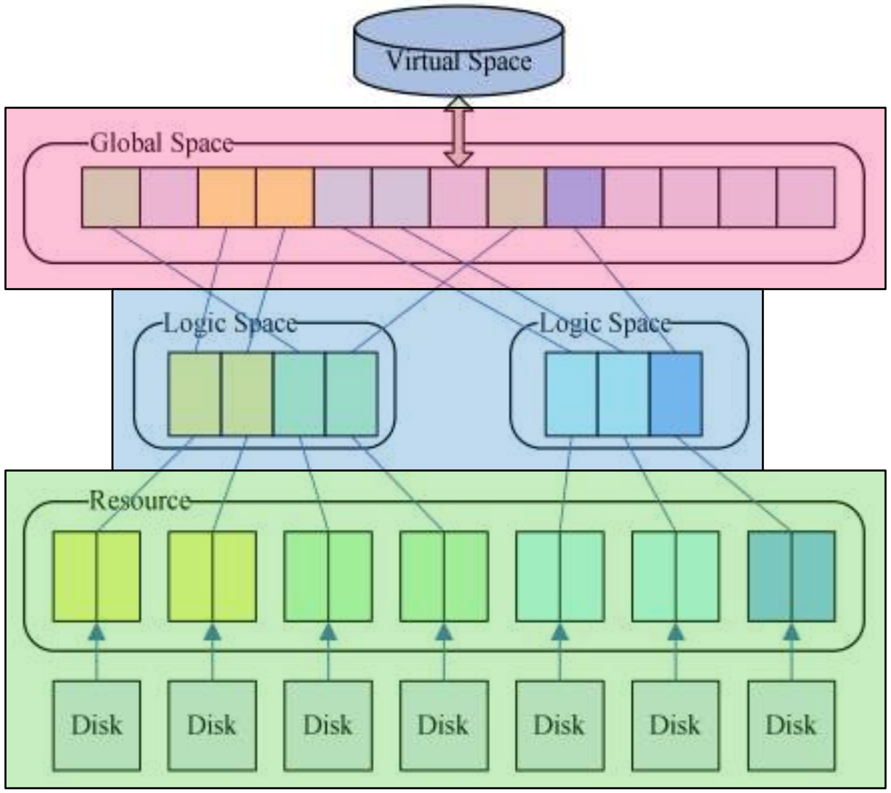
- Introduction
- Load Balancing Based Virtual Storage Architecture
    - Storage Virtualization Model (SVM)
    - Virtual Storage Architecture (VSA)
- Virtual Storage Implementing
    - Replica Balancing Module
    - Writing Balancing Module
- Use Case
- Conclusion

Introduction

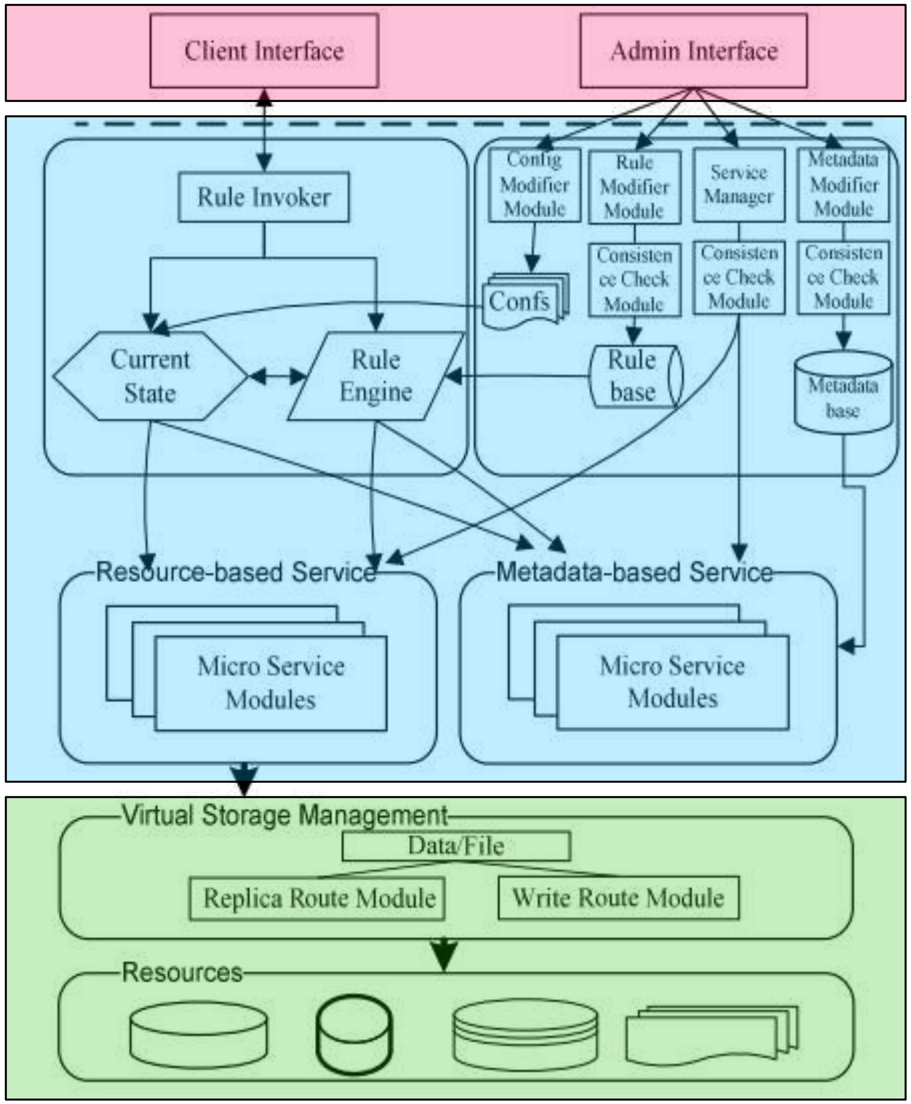
- The exponential growth of data needs higher performance data storage. But local storage strategy still has some lack of capability.
- Cloud Storage is an important part of Cloud Computing, and it provides a way to achieve large scale and cheaper storage architecture.
- This paper proposed a load balancing virtual storage strategy (LBVS) and use Fair-Share Replication (FSR) and a writing balancing algorithm to archive it.

Load Balancing Based Virtual Storage Architecture (1/2)

- Storage Virtualization Model (SVM)
  - Resource virtualization
    - Maps physical devices to uniform virtual resource
  - Logical space virtualization
    - Maps uniform resource to basic nodes (collections)
  - Storage network virtualization
    - Assembles collections in logical space into global space



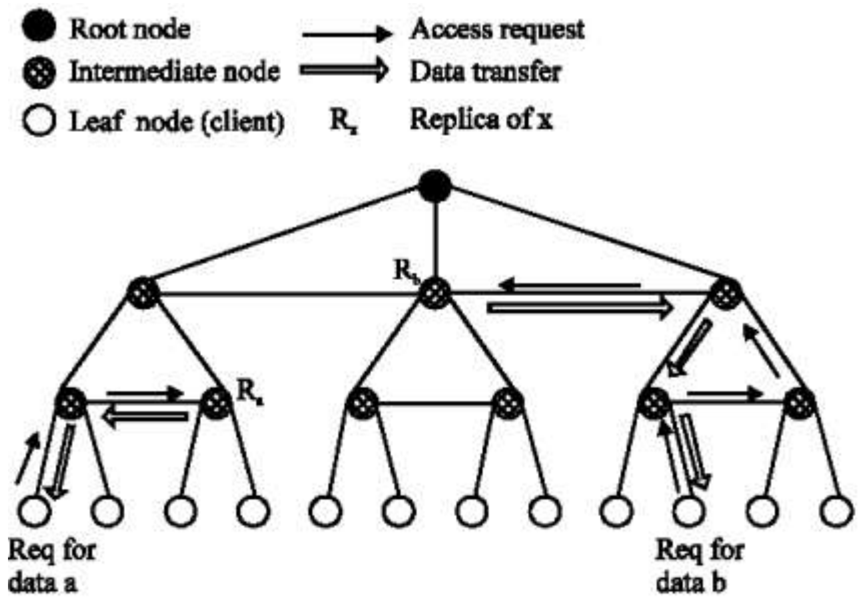
Load Balancing Based Virtual Storage Architecture (2/2)

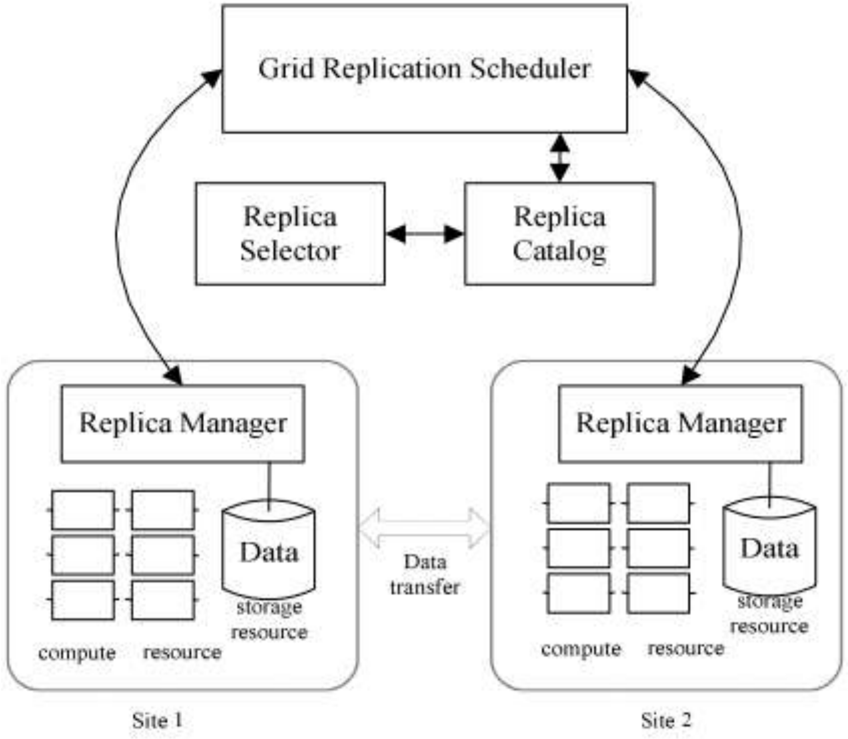


- Virtual Storage Architecture (VSA)
  - Based on SVM
  - Interface Layer
    - icommands
    - Client
    - Web Browser
    - Sharing Folder
  - Rule and Metadata Management Layer
    - Upper layer:
      - Client and Admin
    - Under layer:
      - Resource and Metadata
  - Virtual Storage Management Layer

Virtual Storage Implementing (1/10)

- Replica Balancing Module (1/7)
  - Uses the algorithm: Fair-Share Replication (FSR)
  - Reference: Rasool Q, Li J, Oreku GS, Munir EU (2008) Fair-share Replication in Data Grid



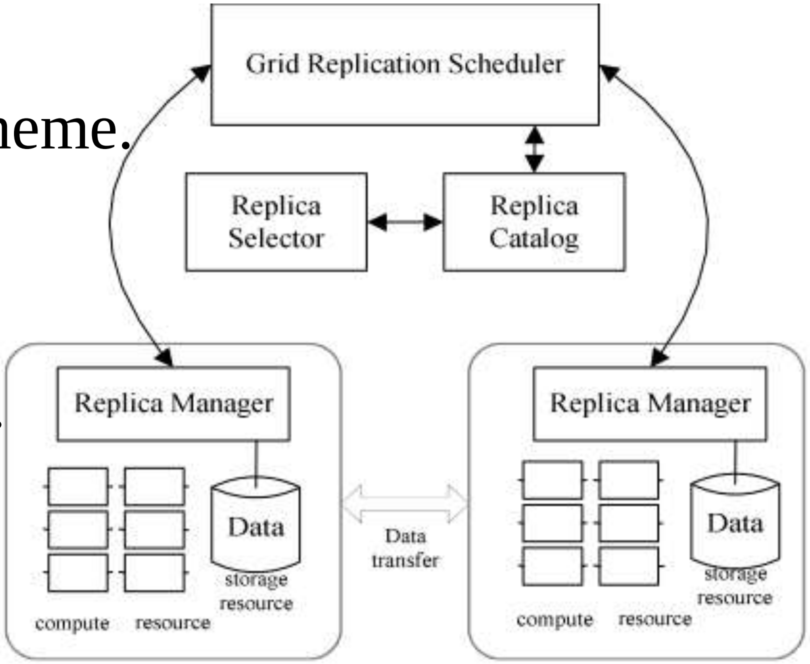


Virtual Storage Implementing (2/10)

- Replica Balancing Module (2/7)
  - To satisfy the latency constraints, there are two ways:
    - Vary the speed of data transfer
    - Shorten the transfer distance
  - Since bandwidth and CPU speed are usually expensive to change, the second way by placing replicas of data objects closer to clients is the cheapest way.
  - The main idea of FSR is to identify best candidate nodes for replica placement primarily based on access load.

Virtual Storage Implementing (3/10)

- Replica Balancing Module (3/7)
  - Grid Replication Scheduler (GRS)
    - The central managing entity for FSR scheme.
  - Replica Catalog
    - Used to register replicas when they are created and placed at the selected nodes.
    - Stores the mapping from the logical file name to the physical file name.
  - Replica Manager
    - Stores information about requested files and the time when requests were made.
    - This information is accumulated and communicated to GRS into a global workload table G(arrival time, clientid, field).



Virtual Storage Implementing (4/10)

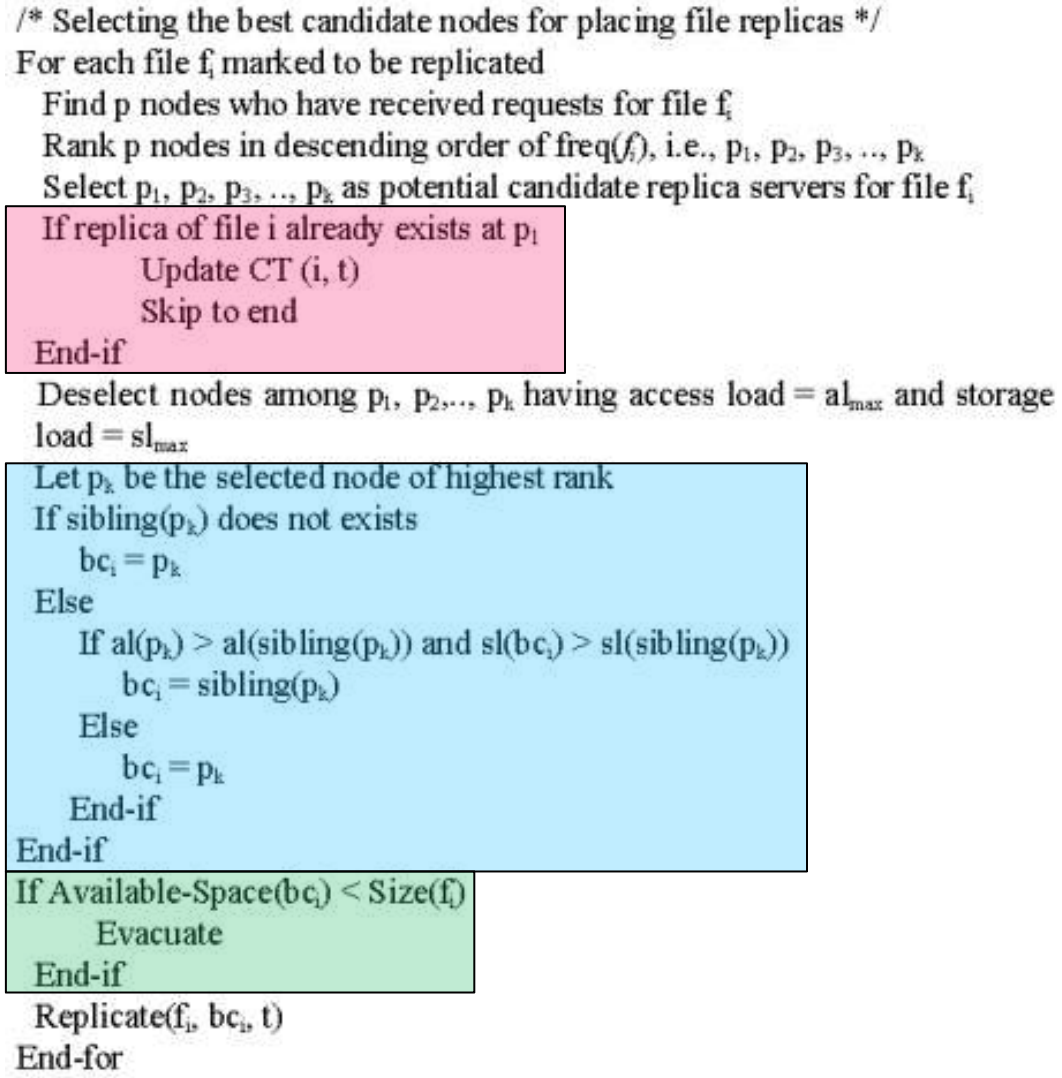
- Replica Balancing Module (4/7)
  - Replica Creation
    - At interval, the table G is processed to get a cumulative workload table W(fieldid, clientid, freq).
    - GRS will calculate the average access frequency (Freq$_{avg}$) from table W.

$$\boxed{Freq_{avg} = \frac{\sum freq}{n}}$$

    - The files which have access frequency greater than or equal to Freq$_{avg}$ are marked for replication.
    - The GRS maintains all the necessary information (creation time, hosting nodes, etc) about the replicas in Replica Catalog.

Virtual Storage Implementing (5/10)

- Replica Balancing Module (5/7)
  - Replica Placement
    - If replica of file i exists:
      - Just update its creation time
    - Determine the best node:
      - Order by freq Desc: Pk … P1
      - If Pk doesn't have siblings, just select Pk as place node.
      - Else compare Pk with siblings. See which one is better. (Access load, Storage load)
    - If best node's available space is not enough, evacuate it.
    - Replicate the file to this node.



```
/* Selecting the best candidate nodes for placing file replicas */
For each file $f_i$ marked to be replicated
  Find p nodes who have received requests for file $f_i$
  Rank p nodes in descending order of freq($f_i$), i.e., $p_1, p_2, p_3, .., p_k$
  Select $p_1, p_2, p_3, .., p_k$ as potential candidate replica servers for file $f_i$
  If replica of file i already exists at $p_1$
        Update CT (i, t)
        Skip to end
  End-if
  Deselect nodes among $p_1, p_2,.., p_k$ having access load = $al_{max}$ and storage
  load = $sl_{max}$
  Let $p_k$ be the selected node of highest rank
  If sibling($p_k$) does not exists
     $bc_i = p_k$
  Else
     If $al(p_k) > al(sibling(p_k))$ and $sl(bc_i) > sl(sibling(p_k))$
        $bc_i$ = sibling($p_k$)
     Else
        $bc_i = p_k$
     End-if
End-if
If Available-Space($bc_i$) < Size($f_i$)
      Evacuate
  End-if
  Replicate($f_i$, $bc_i$, t)
End-for
```

Virtual Storage Implementing (6/10)

- Replica Balancing Module (6/7)
  - Replica Selection
    - We use closest policy (Benoit et al., 2007) for replica selection
    - Replica Selector is responsible to implement this service:
      - On receiving a request for a specific data file,
        the replica selector query replica catalog to get information about all the available replicas.
      - Return the location of the replica that offers the highest transfer speed and is least number of hops away from the requesting client.

Virtual Storage Implementing (7/10)

- Replica Balancing Module (7/7)
  - Replica Replacement Policy
    - Which of the stored replicas should be replaced with the new replica if available space of selected node isn't enough.
    - For a given selected node:

1. Check the creation times of all present replicas.
2. Replicas which were created earlier than current time session and currently not being active or referenced will be moved to the upper node. Because it's fast to replica if they become popular in future.
3. Replicas will be deleted permanently from system if it has not been referenced since last two sessions.

Virtual Storage Implementing (8/10)

- Writing Balancing Module (1/3)
  - Uses weight-computing method to get the best node
    - First sets a initial weight IW(Ni) for every node Ni.
    - Module will adjust IW(Ni) dynamically by gathering the following parameters periodically form storage nodes.
      - CPU % Utilization: CPU(Ni)%
      - Memory % Utilization: Memory(Ni)%
      - NetFLow: T(Ni)
      - Disk I/O Frequency: IO(Ni)
      - Response Time: Rt(Ni)
      - Process Sum: Pr(Ni)
    - Adds weight parameter πi to adjust the proportion of parameters.

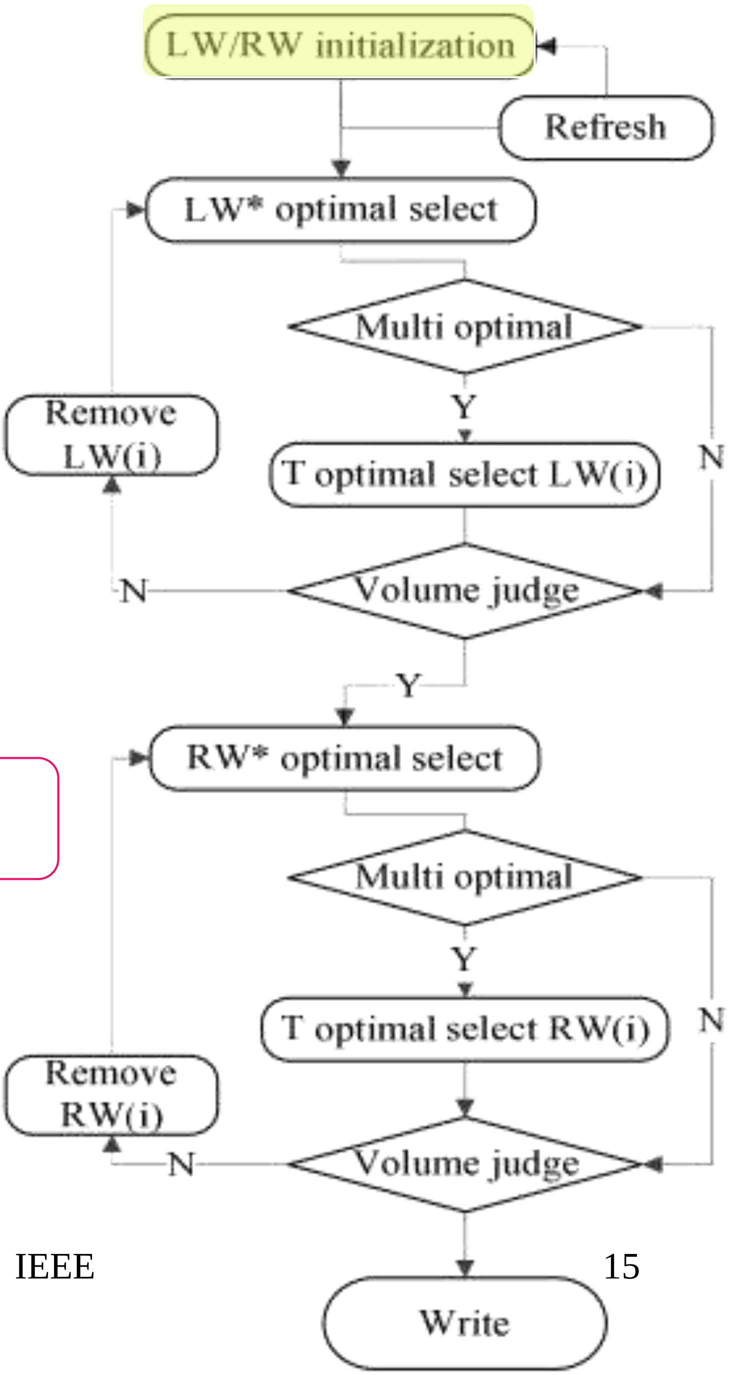
Virtual Storage Implementing (9/10)

- Writing Balancing Module (2/3)
  - $\Sigma \pi i = 1$
  - Load(Ni) = π1 * CPU(Ni)% + π2 * Memory(Ni)% + π3 * T(Ni) + π4 * IO(Ni) + π5 * Rt(Ni) + π6 * Pr(Ni)
  - Refresh Cycle T(i) is bout 5-10 seconds.

Virtual Storage Implementing (10/10)

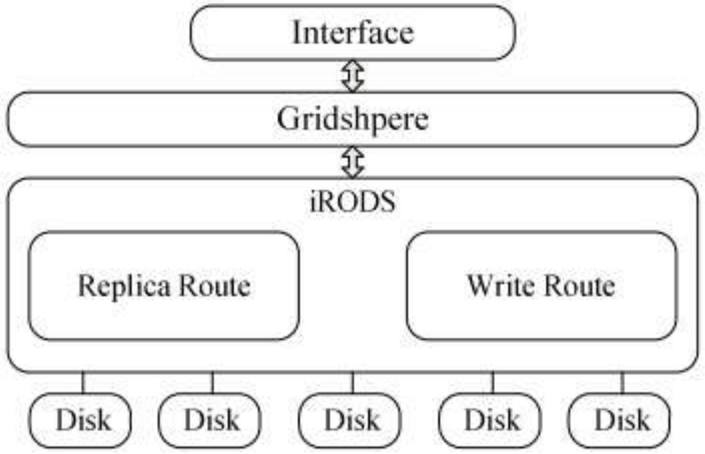
- Writing Balancing Module (3/3)
  - When data/file is writing,
    Writing Routing Module will select best logic node from IW(Ni)
  - IW contains two tables:
    - LW: Logic Space Weight Table
    - RW: Resource Weight Table
  - Determine the routing
    - Find the best node of Logic Space
      - If more than one, select the best T one
    - Find the best node of Resource
    - Routing finished, write

NetFLow?
Resp. Time?

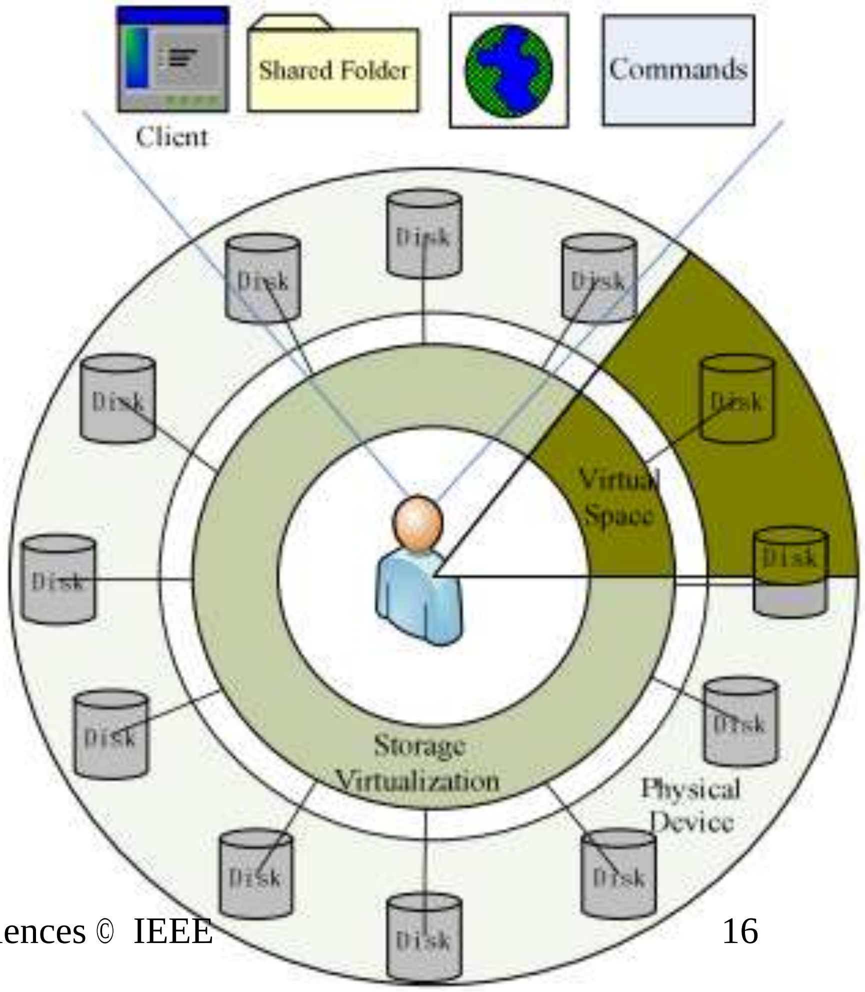


Use Case

- Implementation Model
  - GridSphere Portal Framework (http://www.gridsphere.org)
  - iRODS Data grid system (https://www.irods.org/)



- User View
  - Web browser
  - Command line
  - Application
  - Sharing file folders



Conclusion

- Compared with former distribute storage system, concurrent access efficiency and use rate of storage resource improve obviously after using LBVS having load balancing control.
- In the foreseeable future, cloud storage will get more and more support. And virtual storage will be applied in much larger scale.