Big Data Infrastructure Technologies for Semiconductor Wafer Fabrication Foundries

Hung-Chang Hsiao (蕭宏章)
Professor
Computer Science and Information Engineering
National Cheng Kung University

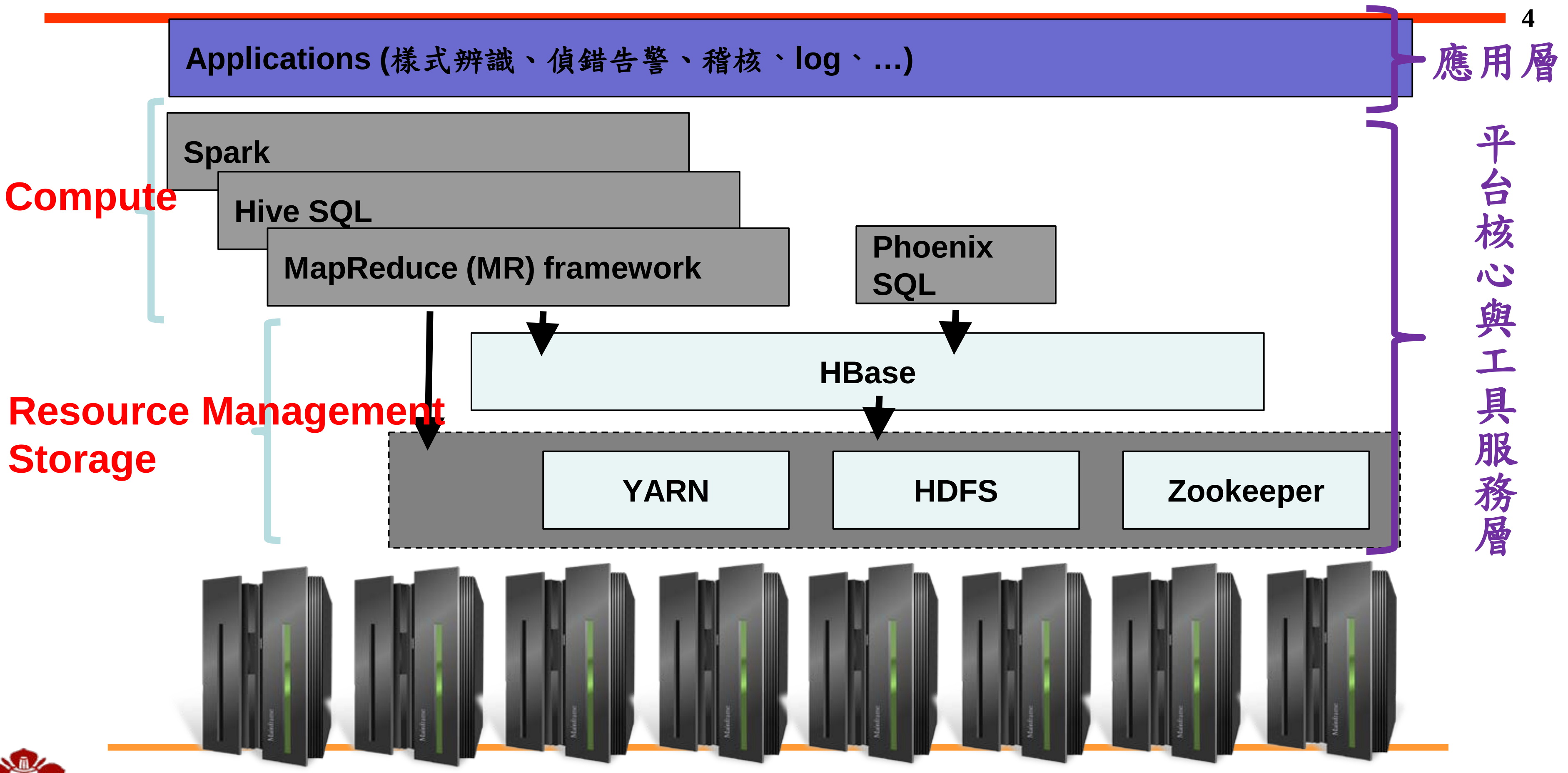
Outline

- 現況與需求
- 運用Open Source開發大型分散式系統基礎設施服務
  - Hadoop Data Service (TDS)
  - Distributed R/Python Computing Service (DPS)
- Status

Outline

- 現況與需求
- 運用Open Source開發大型分散式系統基礎設施服務
  - **Hadoop Data Service (HDS or TDS)**
  - Distributed R/Python Computing Service (DRS or DPS)
- Status

Providing 巨量資料平台 to End Users…



現象 (應用端)

- **分析端的使用者**
  - **眾多相異儲存體**：比如Samba, FTP, Oracle, Microsoft SQL server，以及近年新興的Hadoop資料庫與檔案系統…
  - **操作繁瑣**：相異storage彼此之間搬動、複製資料，甚或夾帶 ETL (extract, transform and load)
  - **學習曲線長**：不易滿足效率要求，尤其面對平行與分散式系統面之技術門檻

現象(平台服務提供與管理端)

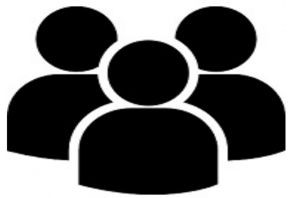
- **管理端的使用者**
  - **新技術的引進未見成效**：舊有的儲存體不敷所需，升級所費不貲，升級後又未必能彰顯績效
  - **新技術之學習門檻**：新興但容易取得的巨量資料平台元件眾多，無法短期一一深入了解，進而開發其效能
  - **新舊系統銜接**：如何將既有的儲存體資料有效率移轉至巨量資料平台
  - **透明性與無痛接軌**：舊有的分析程式如何能被無縫接軌至新興巨量資料平台進行資料存取與計算
  - **簡易管理**：軟硬體元件隨著平台計算與儲存節點增加或毀損而仍能省力去管理

需求：一套儲存系統橋接舊使用者於新技術

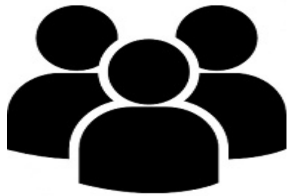
Transparency

Autonomy

Efficiency

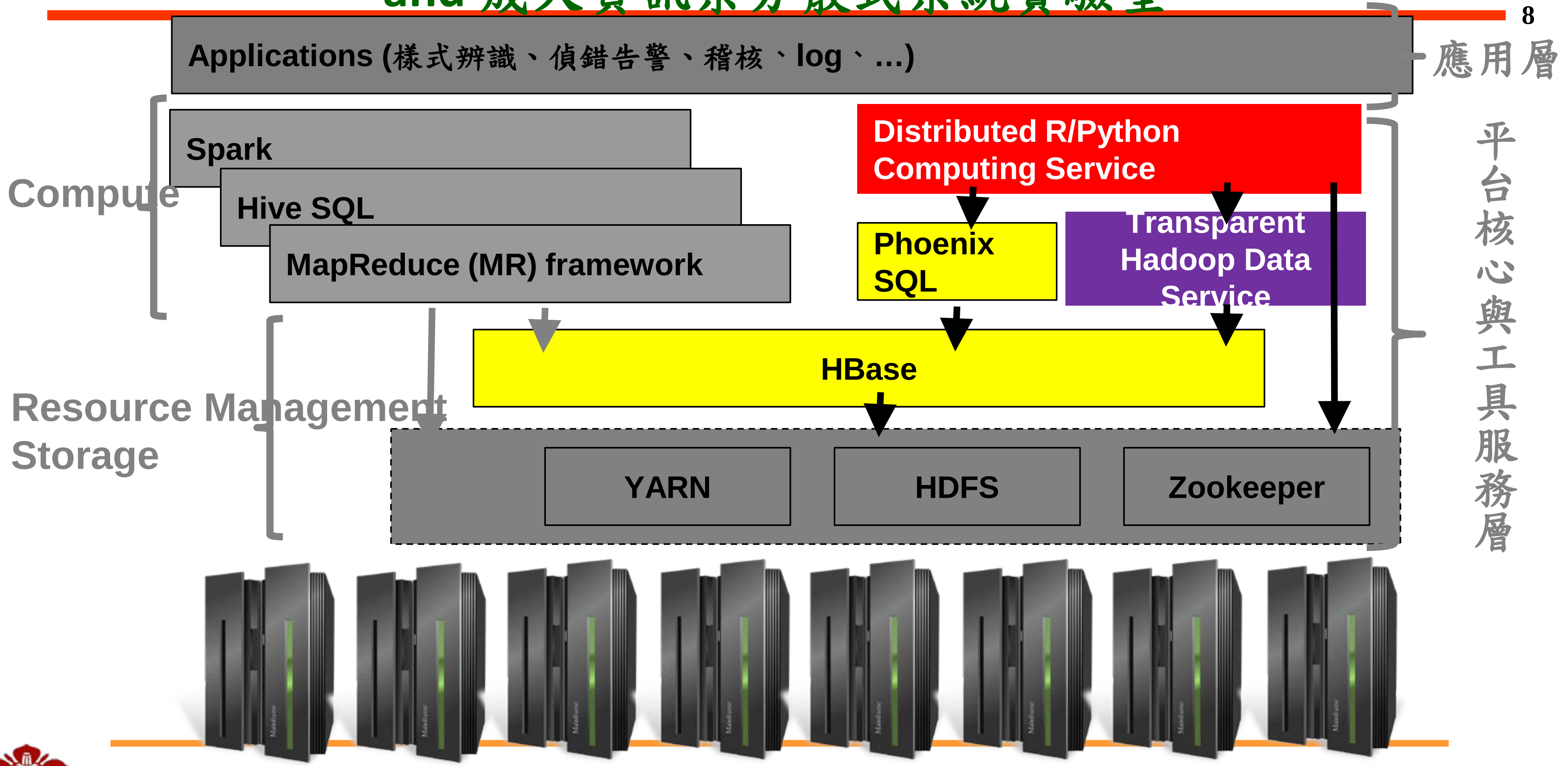


使用者



使用者

Joint Research by Semiconductor Wafer Fabrication Foundry and 成大資訊系分散式系統實驗室


Applications (樣式辨識、偵錯告警、稽核、log、...)
應用層
Compute
Spark
Hive SQL
MapReduce (MR) framework
Distributed R/Python Computing Service
Phoenix SQL
Transparent Hadoop Data Service
HBase
Resource Management Storage
YARN
HDFS
Zookeeper
平台核心與工具服務層

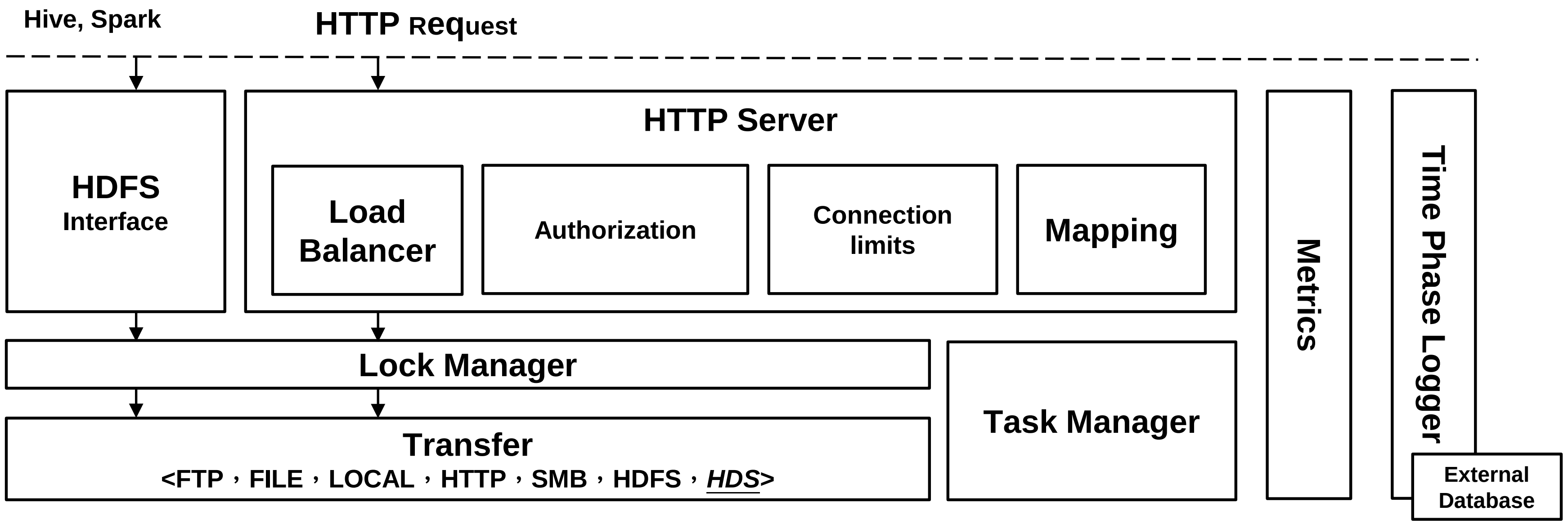
具體成效

- **HDS and DRS has been incrementally developed and operated for 3.5 yrs by 20+ persons (if counting guys in my lab only)**

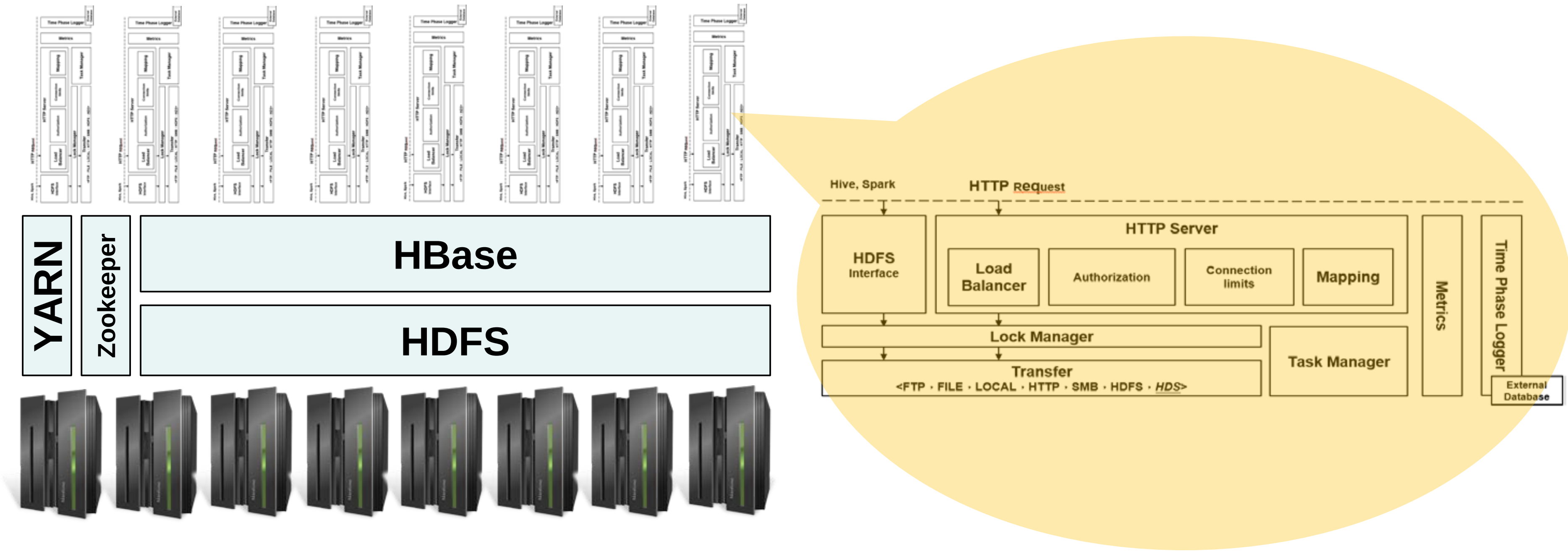
- **With HDS and DRS, 國內產業:**
  - 使某半導體業者提升部分產品良率至少1%，縮短找出製造瑕疵時間從2~3週降至1日內
  - 關鍵原因:
    - 巨量資料量
    - 高速計算

- **國際：**
  - Apache HBase committer and PMC

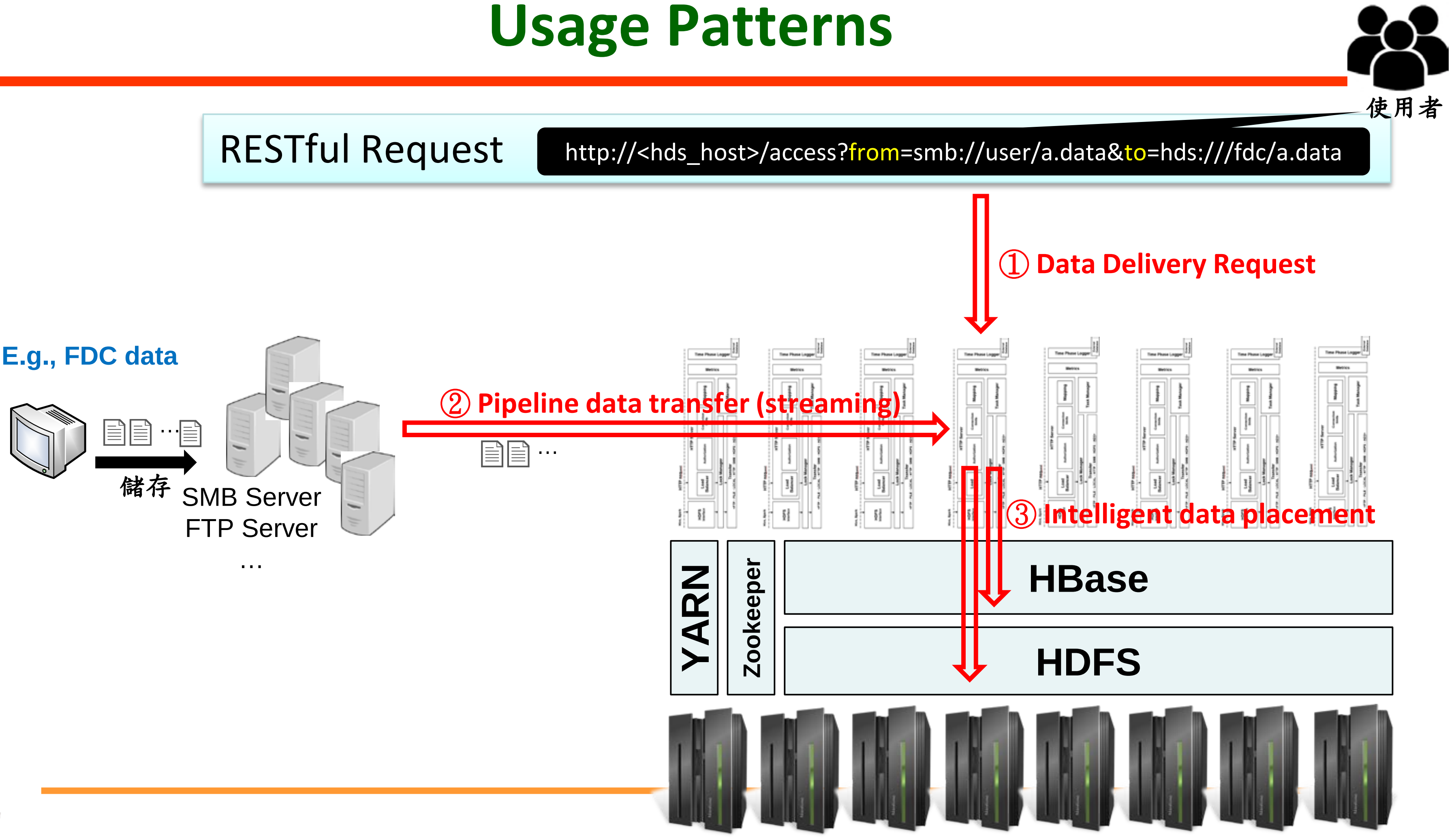
TDS Software Stack



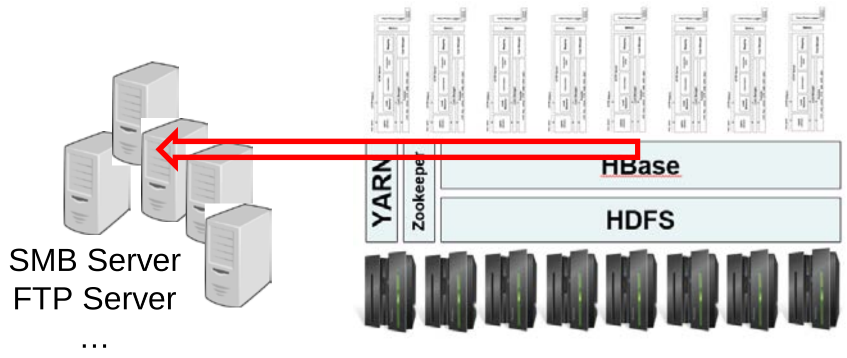
Software Stack: Everything Together

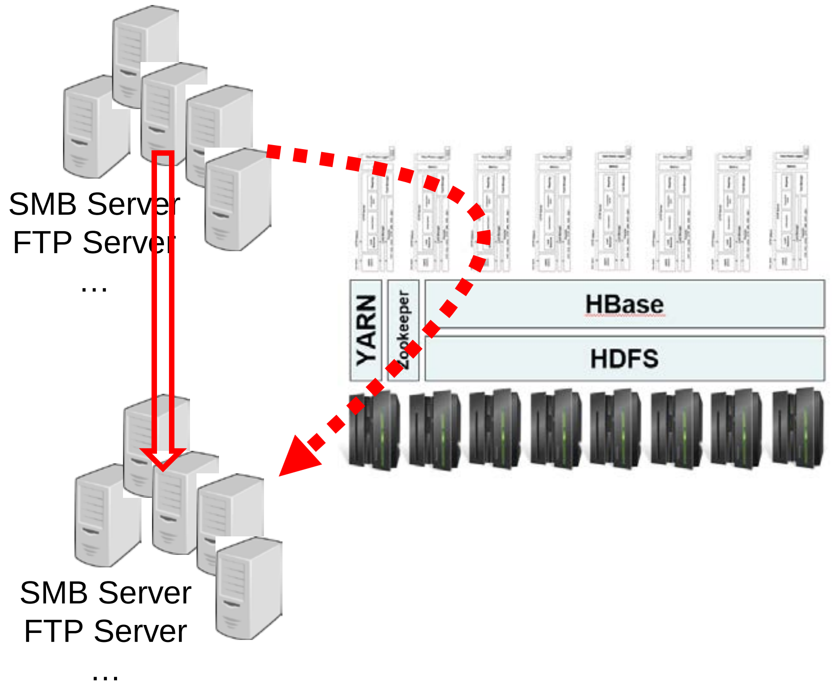


Usage Patterns


使用者
RESTful Request
http://<hds_host>/access?from=smb://user/a.data&to=hds:///fdc/a.data
① Data Delivery Request
E.g., FDC data
儲存
SMB Server
FTP Server
…
② Pipeline data transfer (streaming)
③ Intelligent data placement
YARN
Zookeeper
HBase
HDFS

Usage Patterns (cont'd)

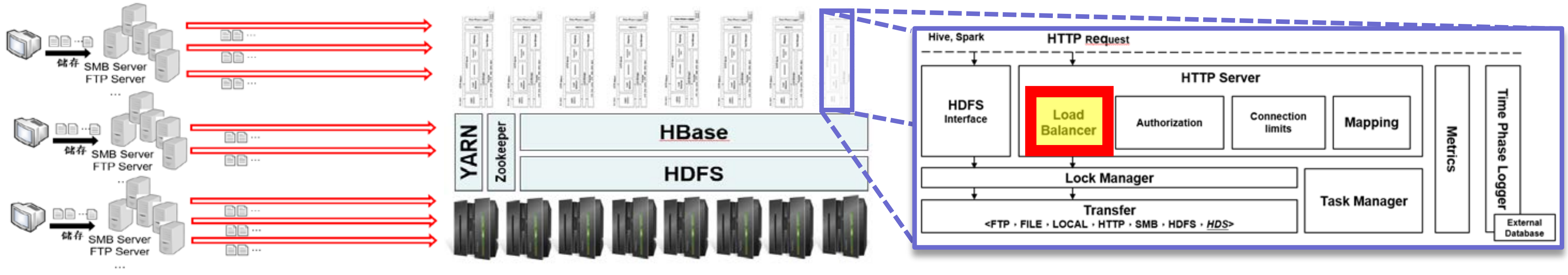




TDS RESTful APIs

- Same APIs in HDS Ver. 1.0 and 2.0
  - `access`
  - `list`
  - `delete`
  - `batchdelete`
  - `watch`
  - `kill`
  - `addmapping`
  - `listmapping`
  - `deletemapping`
  - `loading`

Load Balancing



- Balance loads among HDS servers

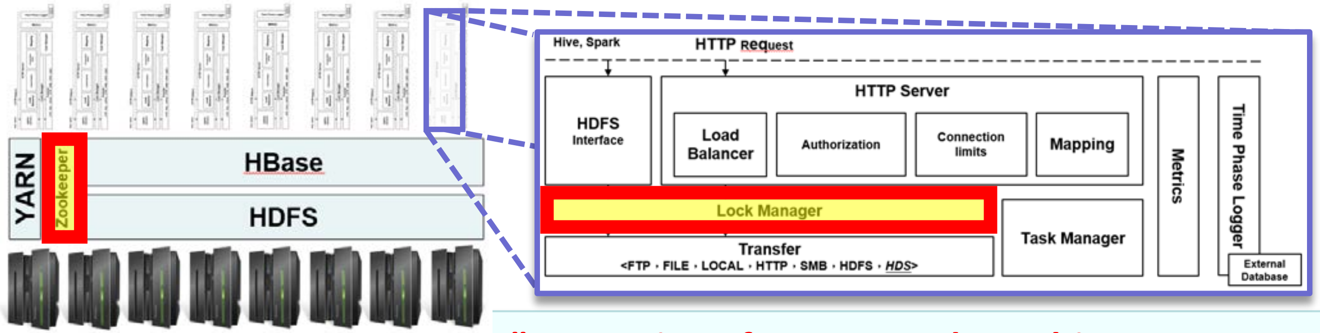
Dynamically and evenly dispatch data transfer requests to HDS servers

On-line probabilistic algorithm: gather loads of servers, and then reallocate requests

H.-C. Hsiao and C.-W. Chang. A Symmetric Load Balancing Algorithm with Performance Guarantees for Distributed Hash Tables. *IEEE Transactions on Computers*, 62(4):662–675, Apr. 2013.

H.-C. Hsiao, H.-Y. Chung, H. Shen, and Y.-C. Chao. Load Rebalancing for Distributed File Systems in Clouds. *IEEE Transactions on Parallel and Distributed Systems*, 24(5):951–962, May 2013.

Lock Service based Sequential Consistency Memory Model

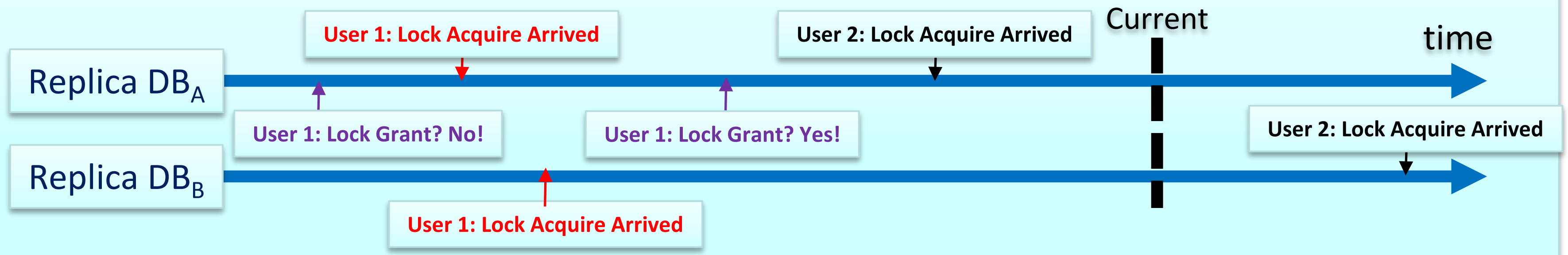


- Serialize concurrent “read” and “write” operations for an HDS data object
- ZooKeeper: replicated databases based on sequential memory consistency model

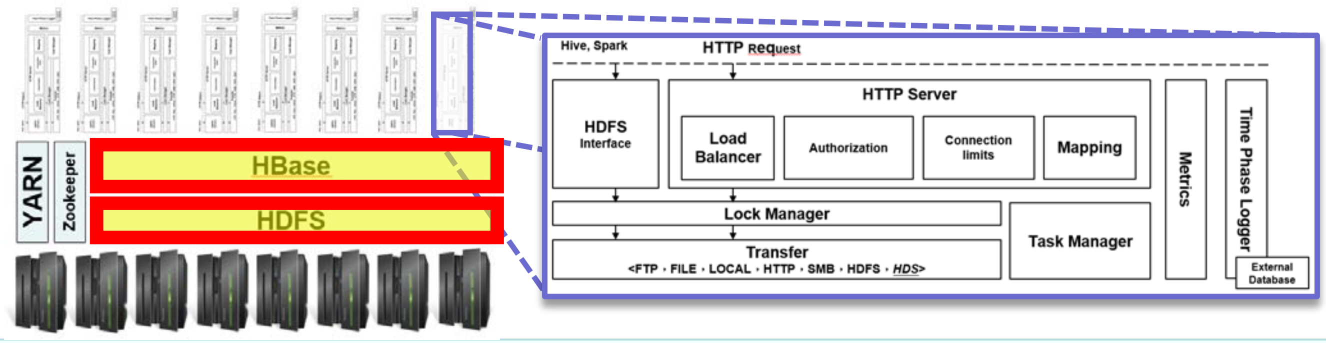
Reads are through local replica

Muti-Paxos writes: serialize write operations to all replicas

- Our locking service:


User 1: Lock Acquire Arrived
User 2: Lock Acquire Arrived
Current
time
Replica DB$_A$
User 1: Lock Grant? No!
User 1: Lock Grant? Yes!
User 2: Lock Acquire Arrived
Replica DB$_B$
User 1: Lock Acquire Arrived

Transparency by Mixing HDFS and HBase



- 每小時約有數十萬個 (100,000x) 小於10 Kbytes的檔案輸出需存到Hadoop
- Hadoop HDFS一個inode大小約150 bytes，則：

每小時在namenode需要的inode空間是x*100,000*150 = 15x Mbytes

每天需15x * 24 = 360x Mbytes

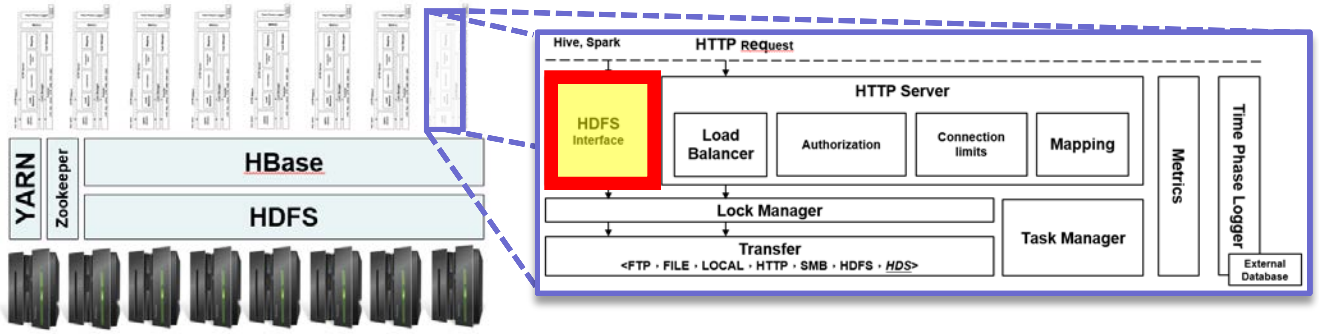
每月需360x * 30 = 10.8x Gbytes

每年需10.8x *365 = 3.942x Tbytes

- iNodes are cached in memory
- Our solution: store small data objects in Hbase (Hadoop distributed database storage engine)

Mutiple small data objects (< 10 Mbytes) are stored in HDFS as a single file, i.e., Hfile (> 64 MBytes); store in HDFS, otherwise

In Compliance with HDFS Interfaces, and Seamless Integrated with Hadoop Ecosystem



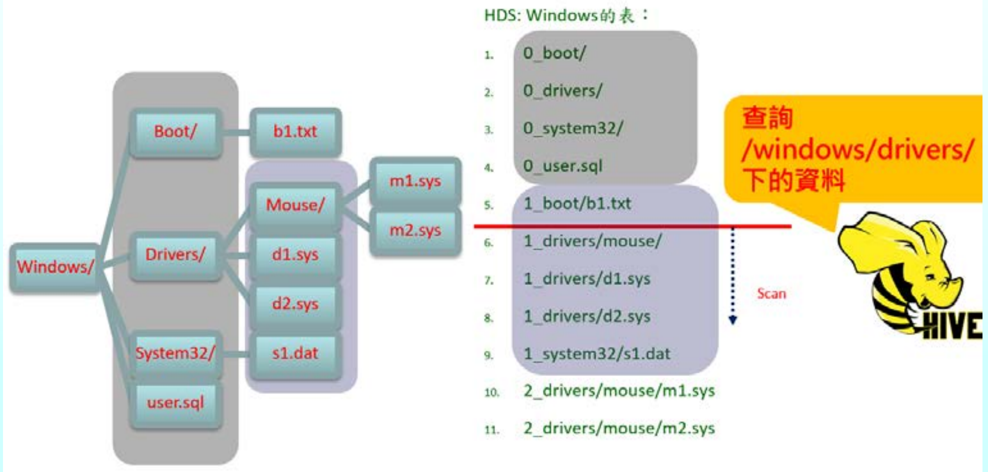
- Compliance with HDFS APIs

Append, create, delete, getFileStatus, getUri, getWorkingDirectory, listStatus, mkdirs, open, rename, setWorkingDirectory, getScheme, initialize,…

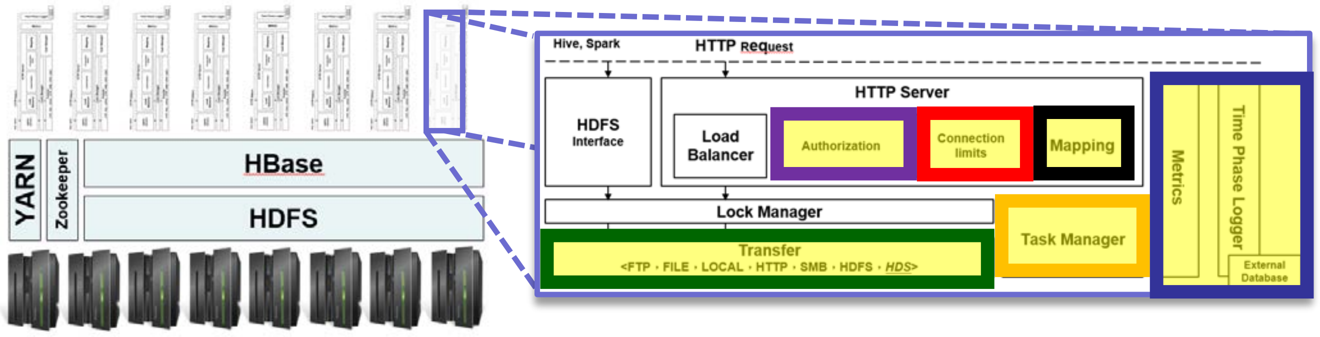
- Directory name space

With HBase to handle directory operations

e.g., Hive directory operations

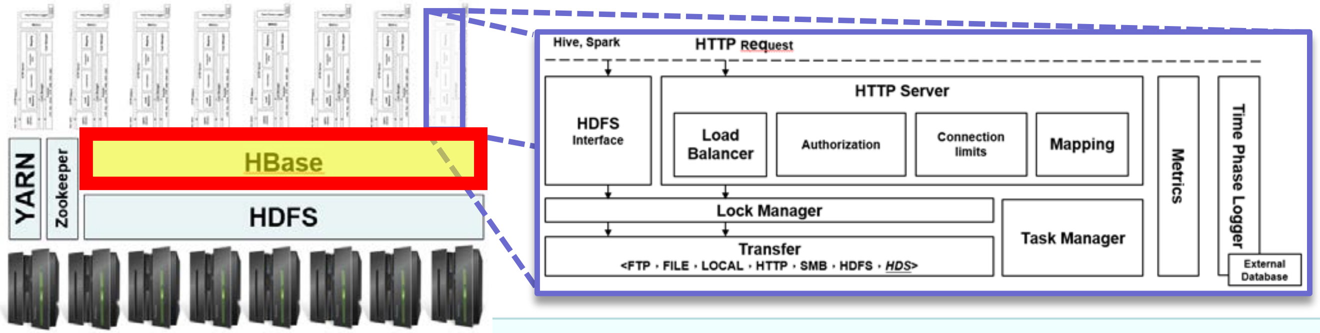


Compliance with HDFS Interfaces, and thus Hadoop Ecosystem



- Service differentiation: long (e.g., `copy`) and short (e.g., `list` and `delete`) RESTful tasks, each with its dedicated connection pool
- Task manager: track and query each task status
- Role-based authorization
- Storage protocols and pipeline streaming: FTP, FILE, LOCAL, HTTP, SMB, HDFS, and HDS, currently
- TinyURL mapping
- Performance metrics: time elapsed/phase and resources for handling a request

Apache HBase Client Side Enhancement

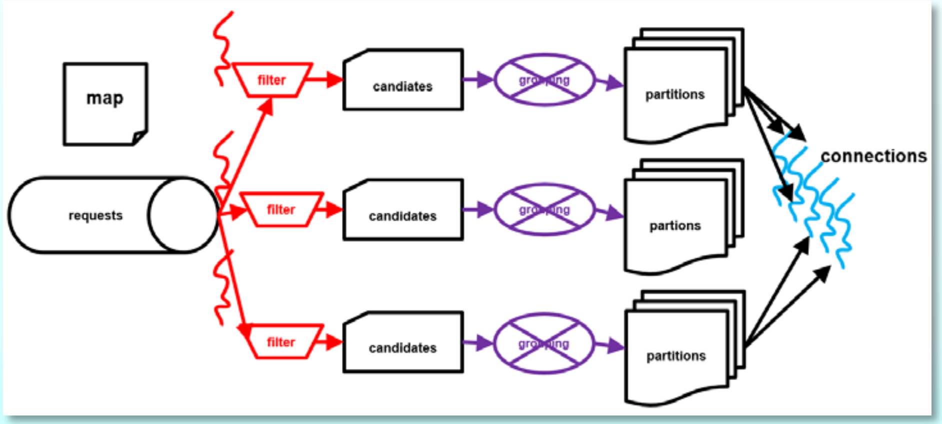


- Write: update, insert and delete (major traffic to HBase)
- Given:

map: regions allocated to region servers
m: maximal number of connections issued by a client
n: maximal number of connections accepted by a region server
t: maximal duration (or timeout) of a connection

- Problem:

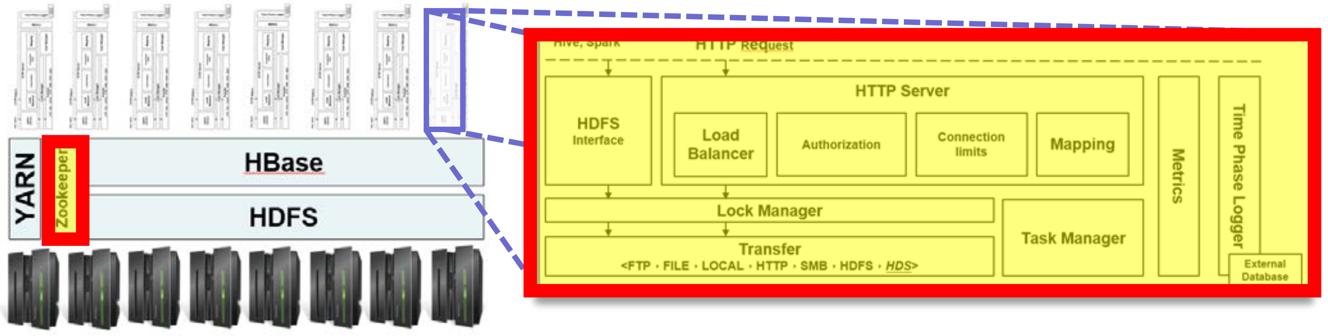
for each client within a time window w, maximize number of successful requests



- **Contribute to Apache HBase 1.3 and 2.0**

Improvements: 20% latency reduction on average per request, using 50% of connections
Implications: 50% free connections for short messages such as read

Scalability and Fault Tolerance



- Each HDS server learns HDS membership on-the-fly
- Read-dominant workload
- Membership information need not be precise and up-to-date
- Based on ZooKeeper

實驗環境

- **Server規格**
  - 型號：Supermicro X80BN
  - CPU：Intel Xeon E7-8850 @2GHz
  - 記憶體：512GB
  - 硬碟：750GB * 16
- **虛擬機配置 (16節點)**
  - CPU：80 x 2GHz
  - 記憶體：32GB
  - 硬碟：750GB
- **叢集版本**
  - Hadoop 2.6.0-cdh5.10.0
  - HBase 1.2.0-cdh5.10.0
  - ZooKeeper 3.4.5-cdh5.10.0
  - Yarn 2.6.0-cdh5.10.0
  - Hive 1.1.0-cdh5.10.0

Benchmark in our development environment, avoiding to introduce extra workloads and traffics to production systems

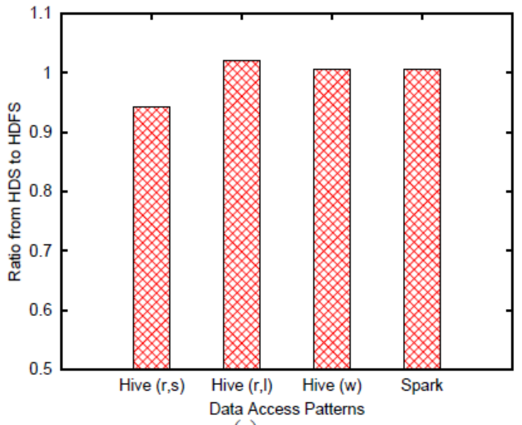
Performance Results

Transparency

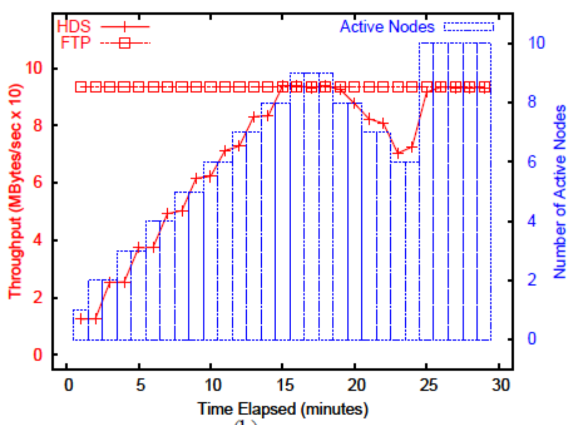
Autonomy

Load balancing

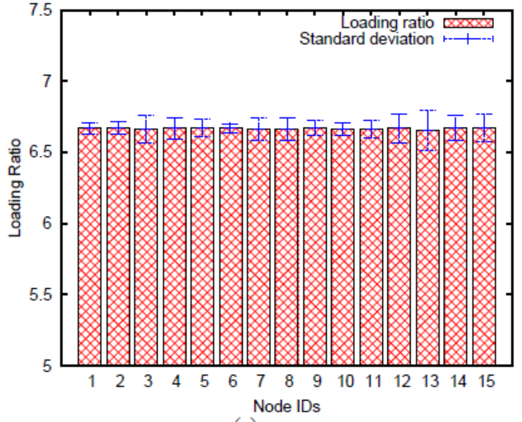
Overheads



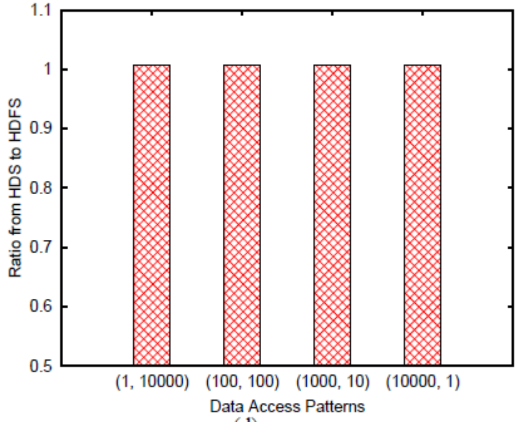
(a)



(b)



(c)



(d)

Related Works

- **SquenceFile and Hadoop archives: Pack small files into a large one**
  - How about metadata, HDFS interface, etc.?
- **WebHDFS and httpFS: HDFS extension over HTTP**
  - Cannot resolve the namenode metadata issue
- **Apache Sqoop: from DB tables to Hadoop**
  - Not for non-DB storages
- **Flume and Kafka**
  - FIFO data streams+Pub/Sub

Outline

- 現況與需求
- 運用Open Source開發大型分散式系統基礎設施服務
  - Hadoop Data Service (HDS or TDS)
  - **Distributed R/Python Computing Service (DRS or DPS)**
- Status

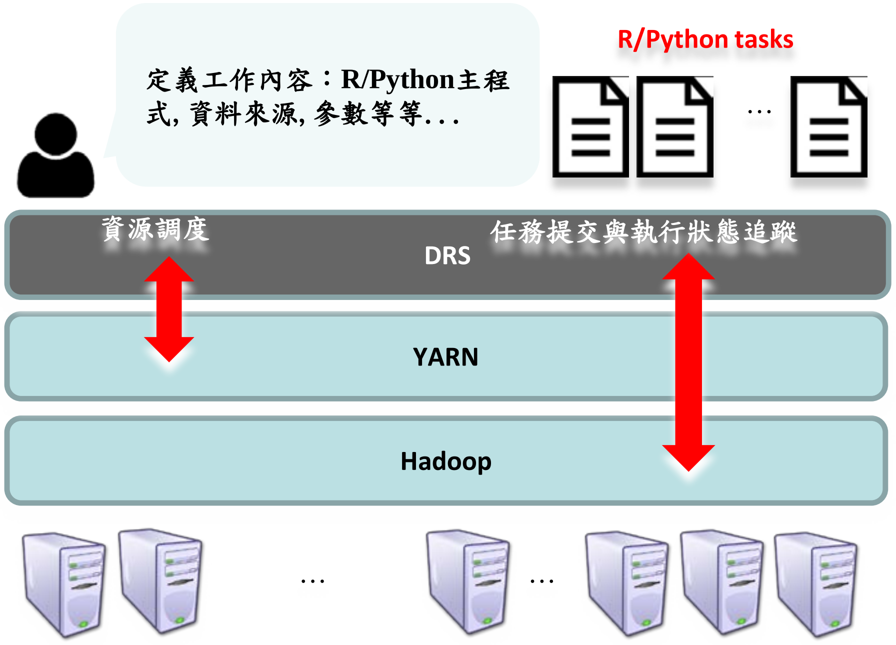
Distributed R/Pythin Computing Service (DPS)

- **Observations:**
  - 大多數分析人員 (非IT背景) 只會願寫單線程程式
  - 該程式普遍有一個pattern：讀資料檔、宣告使用大量記憶體空間，然後計算並輸出
  - 資料量大：生產機器設備生成的資料
- **人員再教育有其難度且成效未必能彰顯**
  - 平行化的複雜度：計算邏輯平行化與資料存取平行化
  - 根據IEEE統計的幾大程式語言: R and Python (是非IT學科會教授的語言)
- **大量資料的計算工具: Hadoop MR及Spark**
  - Need to familiar with MR “framework”
  - 本質議題：平行化 even with Spark
- **已經存在的R/Python程式需因應“新興”平台而重新開發**

DPS: A Quick Overview

自動化的將R/Python程式分散到各個機台執行的服務

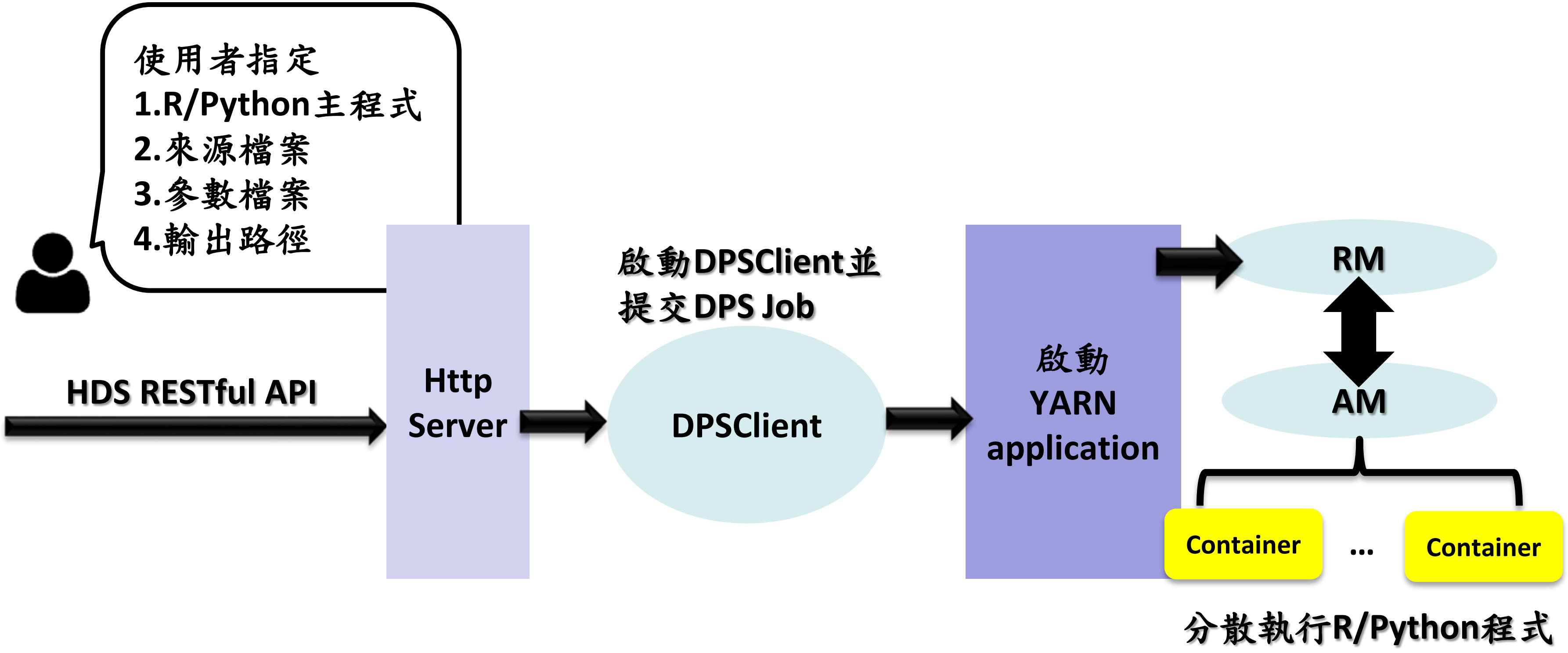
- 使用者自定義工作內容
- 基於Hadoop/YARN的分散式系統環境
- 透過DRS有效率的將任務分散到各機台執行



DPS: A Quick Overview (cont'd)

- 支援相異資料來源，該來源必須被抽象成非IT人員會存取的 with TDS
- **Features:**
  - Batch input: a set of data objects in HDS
  - Fault tolerance: tolerate hardware/software faults while lasting computation
  - Scalability: scale if resources are available
  - Efficient resource management: memory monitoring/allocation/deallocation/sharing among tasks and among jobs
  - Not real-time currently, and for big job only
    - Big job can tolerate startup cost while real-time small job cannot
- **Akin to Hadoop MR framework**
  - Job: AM, NM's (containers), heart beat among AM and containers, tasks scheduling

Internal Workflow



使用者角度 (The Case of R programs)

Configuration:

```
http://host/hds/v1/run?code="…"&data="…"&config="…"&codeout="…"&consoleto ="…" //fork a DRS job
code = … //R program path
data = … //input data file, containing a list of paths of data files
container_spec = … //specification for a set of containers
codeout = … //aggregation of output data files for R programs running in each container
consoleto = … //R program console output
…
```

R program body:

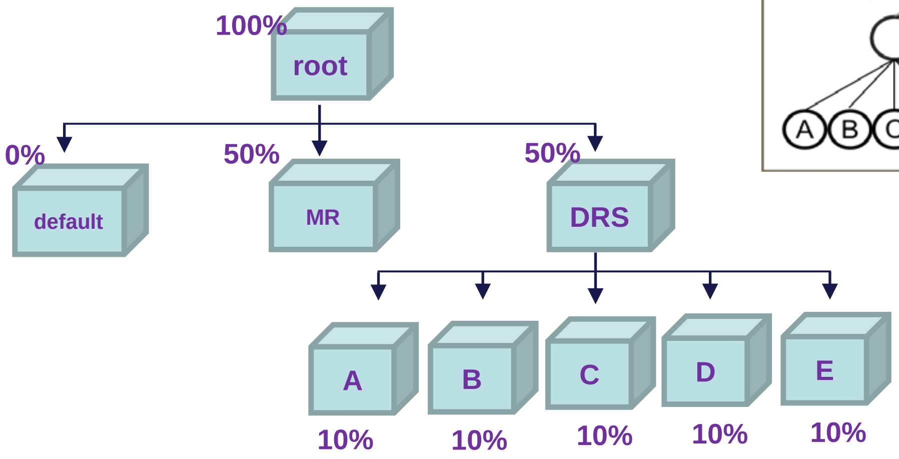
```
tempData <- read.csv(DRS_INPUT_FILE_PATH , header = TRUE) //read data files locally
data <- …(tempData) //filter and load data
…
sumResult <- sum(data) //compute sum
sdResult <- sd(data) //compute standard deviation
medianResult <- median(data) //find median
…
```

Development Items

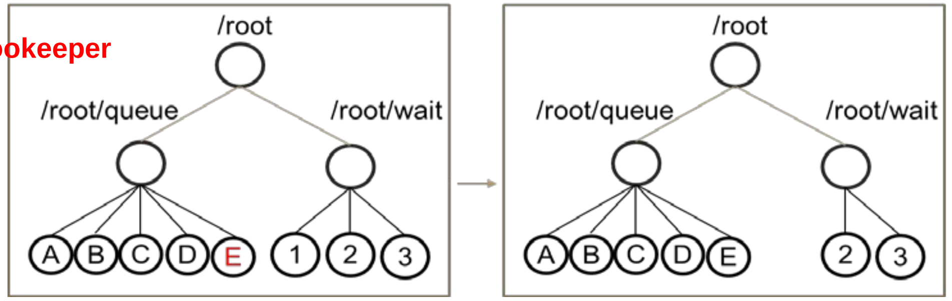
- 多使用者(用戶)資源配置與排程
- For each user,
  - 資源的監控與評測
  - 智慧化的資源調控與配置

管理者角度

YARN Hierarchical Queue



Zookeeper



Queue E 運算結束

順位 1 遞補
進入Queue E

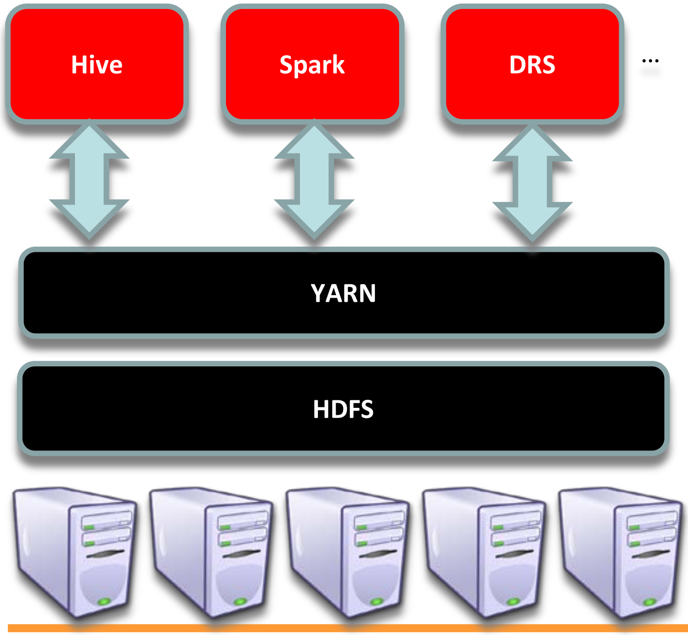
In a Job,…

- **Application master (AM) performs to:**
  - Gather and maintain a list of tasks and a set of resources
    - Per task info: file size, locations, …
    - Per container info: CPU cores, memory, …
  - Match tasks and containers (task scheduling issue)
    - E.g., largest task first and random assignment
  - Dispatch tasks to containers
  - Monitor and gather tasks execution status for performance and failure recovery
  - Even vary resource spec. on-the-fly
  - Overlap scheduling with container computation

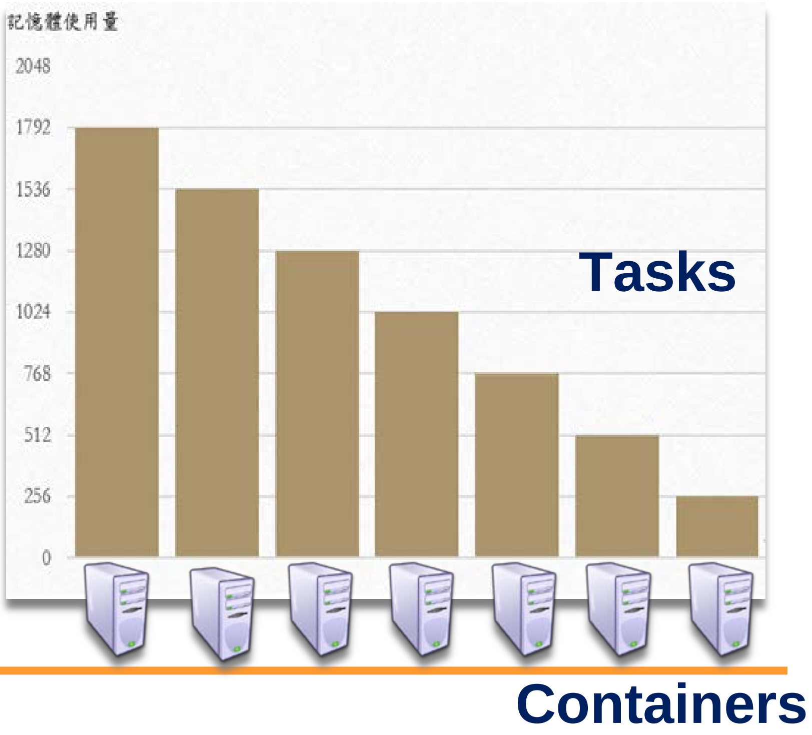
- **"Per" container performs:**
  - To wait for notification of task assignment
  - Then to fetch task data and compute
  - To return AM computation status

Jobs Heterogeneity

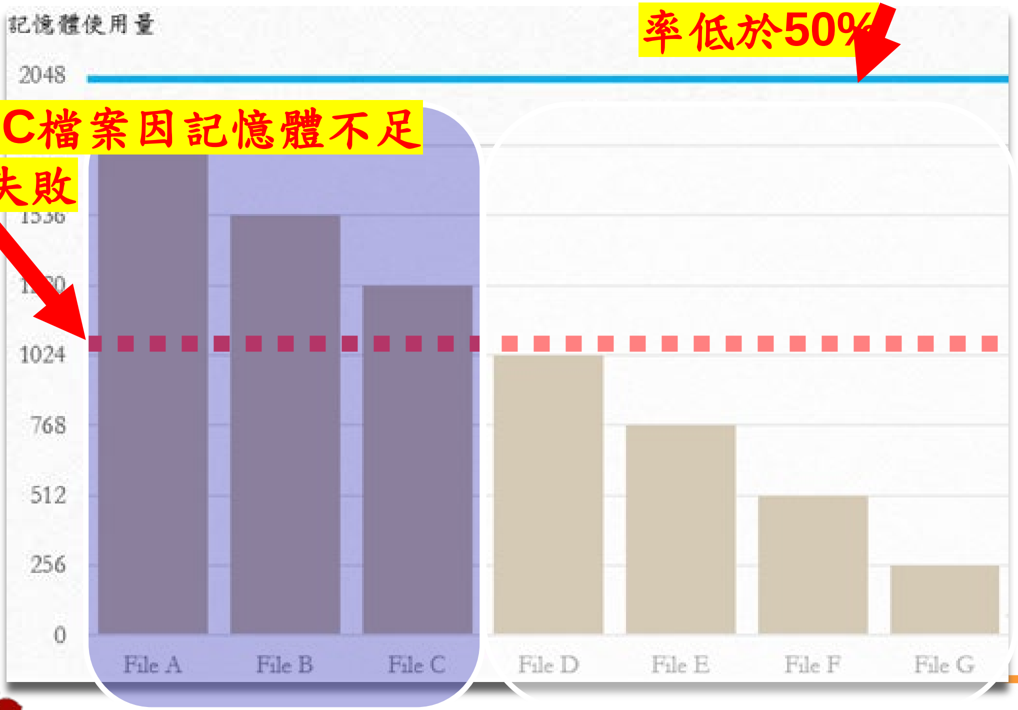
叢集多用戶
(每個用戶的資源皆需要“自我”好好管理)



單一DRS job內部tasks資源需求不一



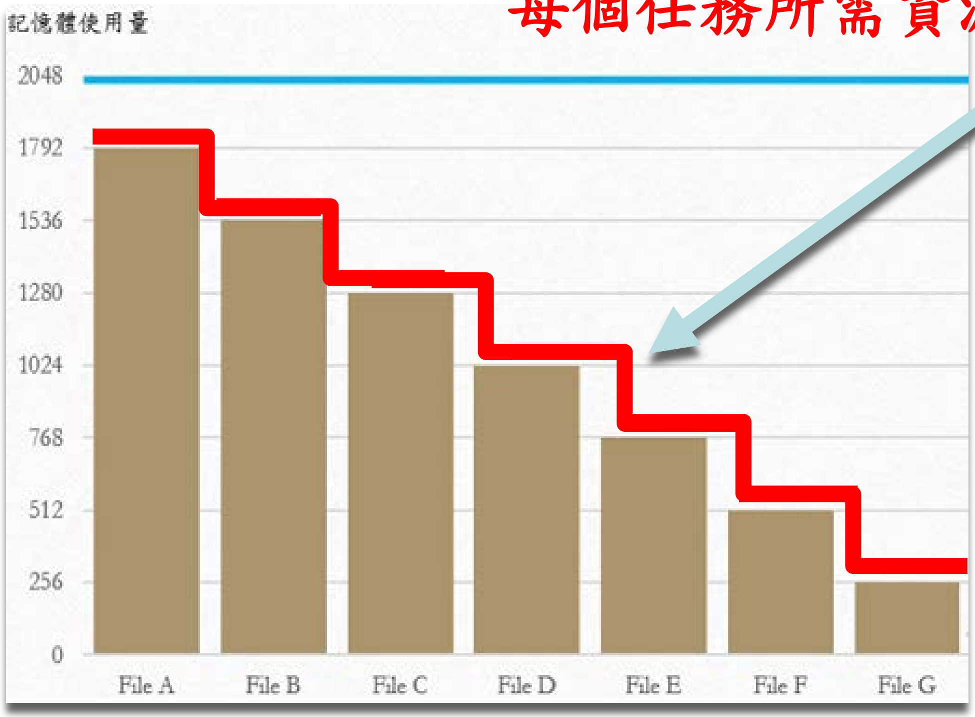
資源過剩與不足



- 初始記憶體設置：
  - 2048 MB: 資源浪費
  - 1024 MB: 資源不足
- 應該多少才適當? 誰決定? 程式開發人員? System operators?
- 須了解程式的“runtime”行為
- Even challenging when resources come and go

目標

理想上：
每個任務所需資源都配得 “剛剛好,” “anytime”

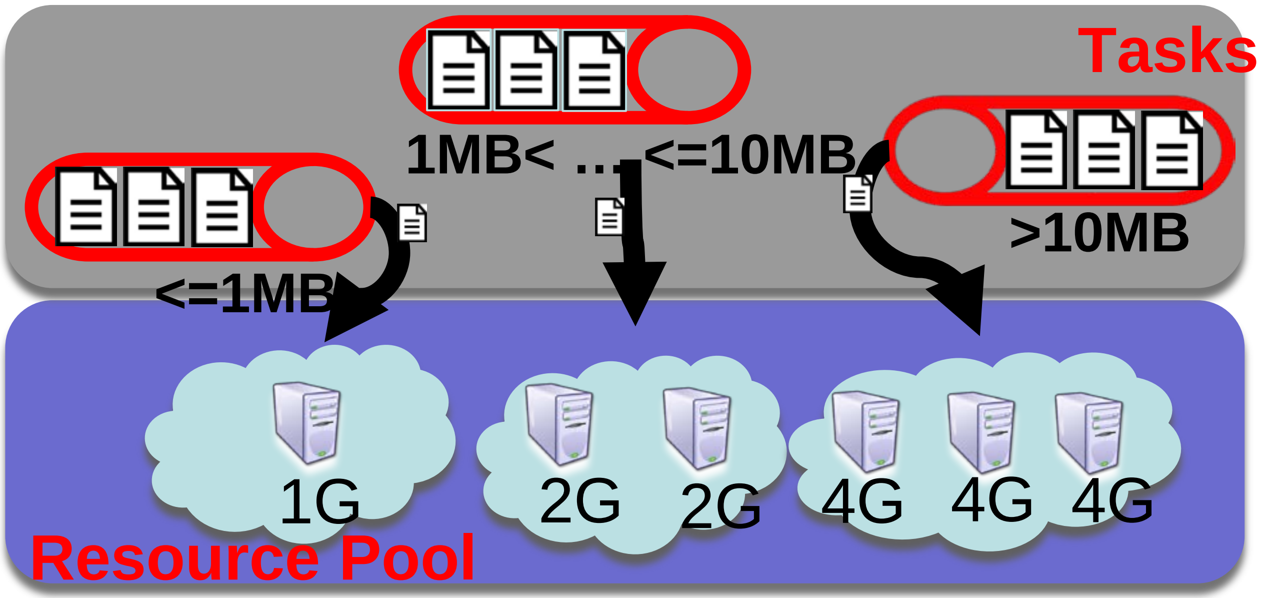


目標：

一方面　　提高任務的成功率

另一方面　降低無必要之資源配置

Solution Architecture



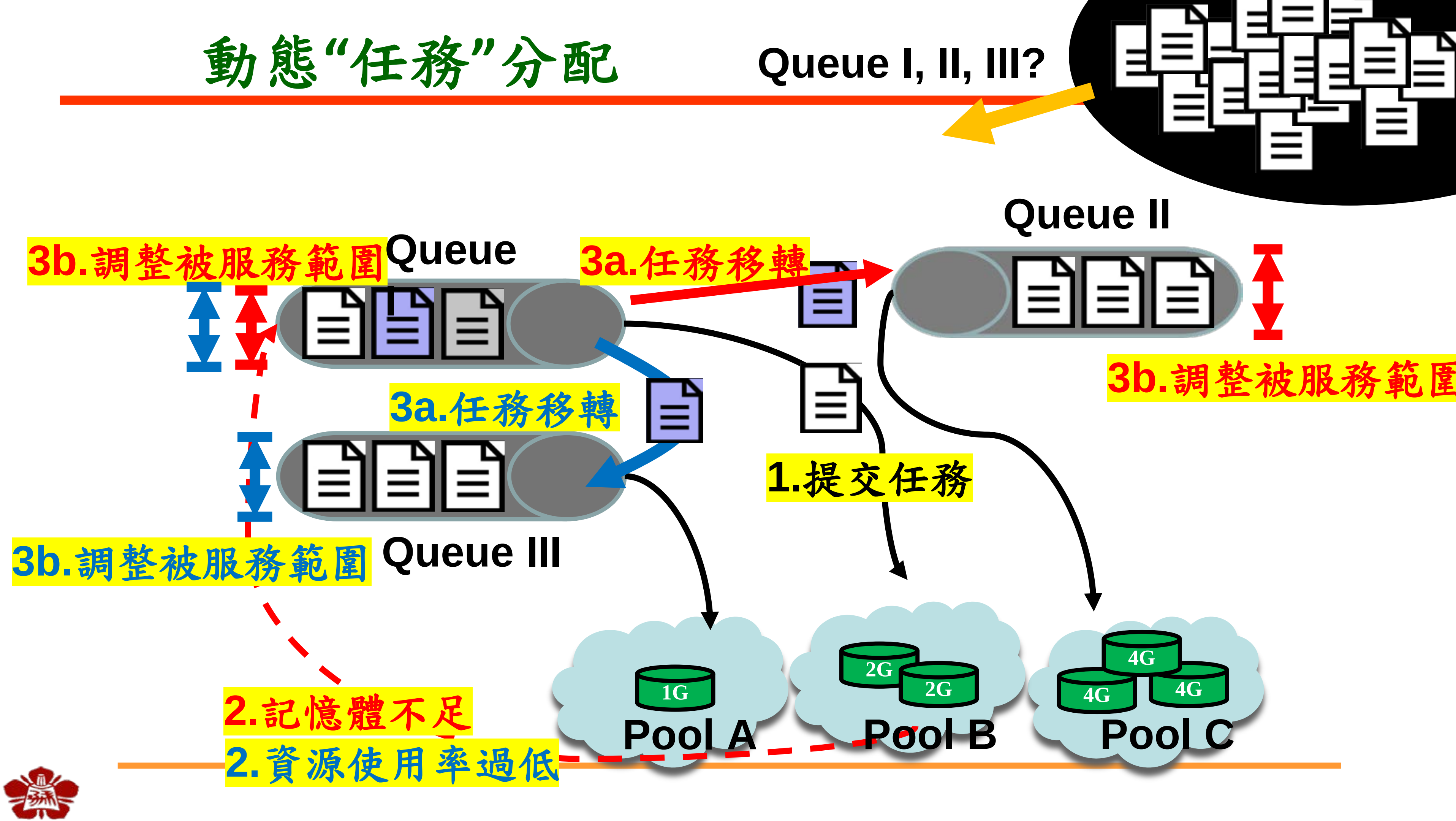
Multiple Queues:
characterized by task attributes (file size)

Resource pool:
grouped by comp. quantity (mem space) per container

Matchmaker:

- learn on the fly by measuring 工作成功率及資源使用量
- matching rule

動態“任務”分配


Queue I, II, III?
Queue II
Queue I
3b.調整被服務範圍
3a.任務移轉
3b.調整被服務範圍
3a.任務移轉
1.提交任務
Queue III
3b.調整被服務範圍
2.記憶體不足
2.資源使用率過低
1G
Pool A
2G
2G
Pool B
4G
4G
4G
Pool C

A Bit Detailed

- **Matching rule:**

(container pool, task queue)

- **Repeat:**

Step 1: For each “given” pool, observe 資源使用率及工作成功率 per task
Step 2: Update matching rule (e.g., attribute of a task queue) due to task migration

- **理論基礎：**
  - Multiple-Server-Multiple-Queue (M/G/k model)
  - Service differentiation

動態“資源”調整

- **Refine attributes of a given resource pool in association rule**
  - E.g., total mem. space in Pool A increased from 16 GB to 32 GB
  - Number of containers in Pool A increases from 8 to 16 (assuming 2GB mem. assigned to each container in Pool A)
  - Consequently, ***parallelism is exploited***, thus reducing job execution time

- **Note: Some pool增加資源表示some other pool釋出資源**
  - E.g., some pool B releases 16 GB of mem. space by killing 4 containers each with 4 GB mem.

- **Proportional allocation (針對那些還有任務的queues) for faireness:**
  - For each queue **Q with tasks**, compute:

$$W(Q) = \text{Queue Length}(Q) \times \text{Per Task Exec. Time Expected}(Q)$$

  - Allocate resources $\propto W(Q)$ to Q

資源調整之彈性機制(時機)

- **Preemptive-based :**
  - 依照參數設定時間，週期性觸發。能提早發現是否資源使用是否不恰當；缺點為會中止正在執行的container，適用於執行運算時間較短的檔案。

- **Non-preemptive-based :**
  - Per-task done：每當有任務結束後觸發，適用於執行時間較長的任務，避免執行一半就被迫中止，而浪費前面運算的時間與所占用的資源。
  - Unused resources：當有一個佇列確定在無任務，則觸發將資源釋出分配給其他區間。

實驗數據(工作執行時間)

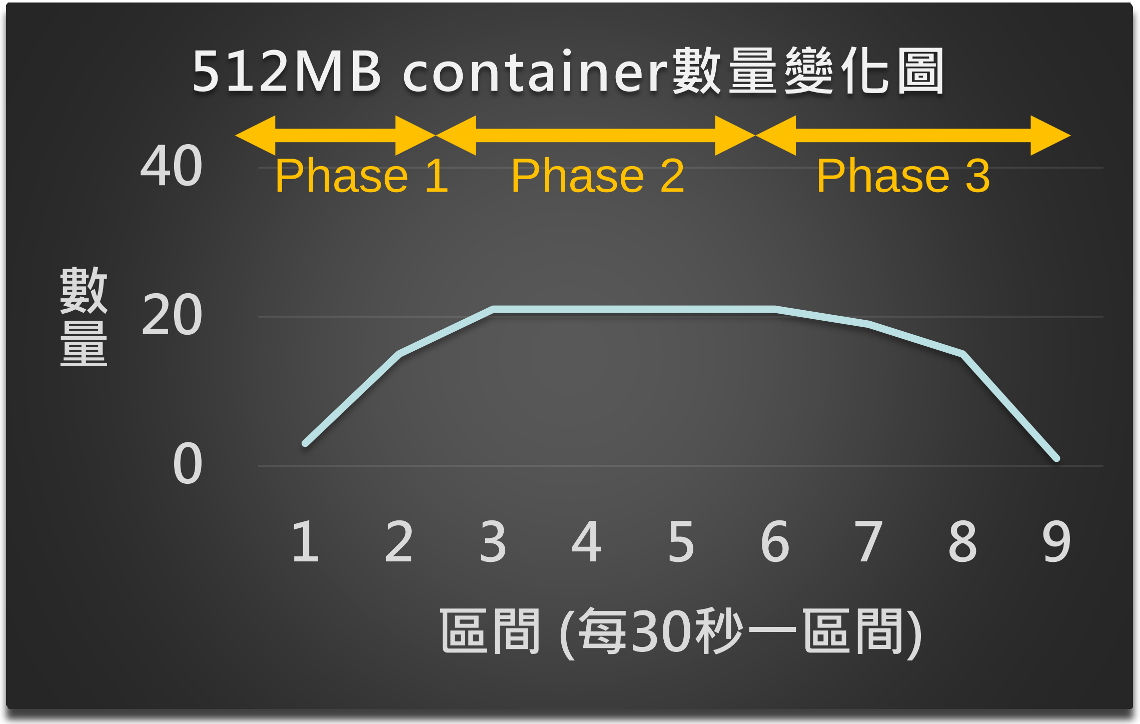
- 情境：
  - 大量的小檔案(約100 KB)與少量的大檔案(10 MB)
  - 1000個任務：有98%是小檔案，2%大檔案
- 資源配置：16GB的總記憶體資源，分成4個pools，其中一個pool具有8個計算單元且每個單元有512 MB的mem，另三個則4個1 GB, 2個2 GB，和1個4 GB
- 週期性動態調整的機制

	DRS 1.0	DRS 2.0
第一次	9分48秒	6分53秒
第二次	9分42秒	6分48秒
第三次	9分53秒	6分50秒

- **觀察：DRS 2.0能有效利用資源，動態調控資源池裡之資源量並將之配置至適當的工作任務，使整體執行時間明顯縮短**

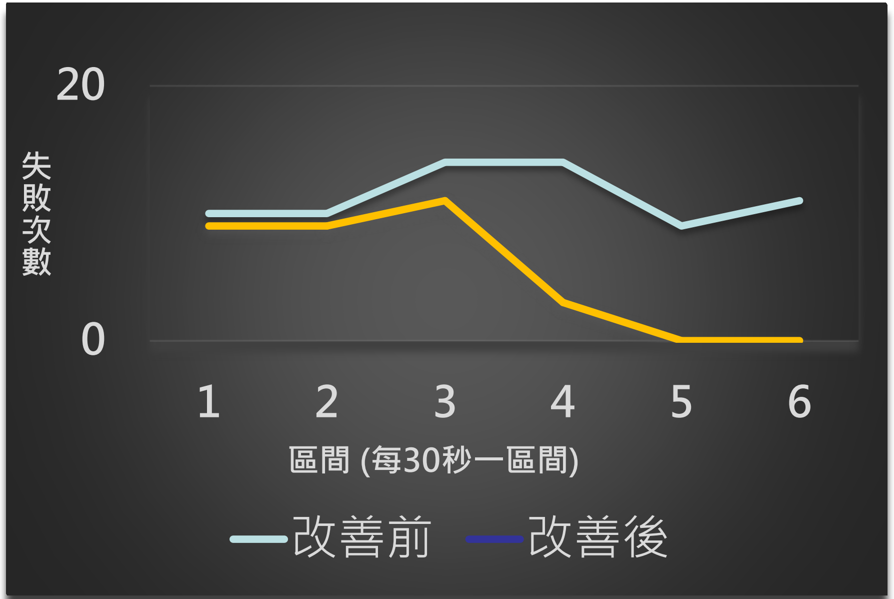
實驗觀察：動態資源調整與配置

- **Phase 1:**
  - 隨著任務執行時間的推進，資源被動態調整使匹配當下計算工作的資源需求
  - 逐漸提升計算的平行度
- **Phase 2:**
  - 此時平行度約20+
- **Phase 3:**
  - 釋放資源
  - 得供其他用戶 (e.g., Spark) 取用



Stress Test (任務成功率)

大量的小檔案 (約100 MB) 與少量的大檔案 (10 GB)



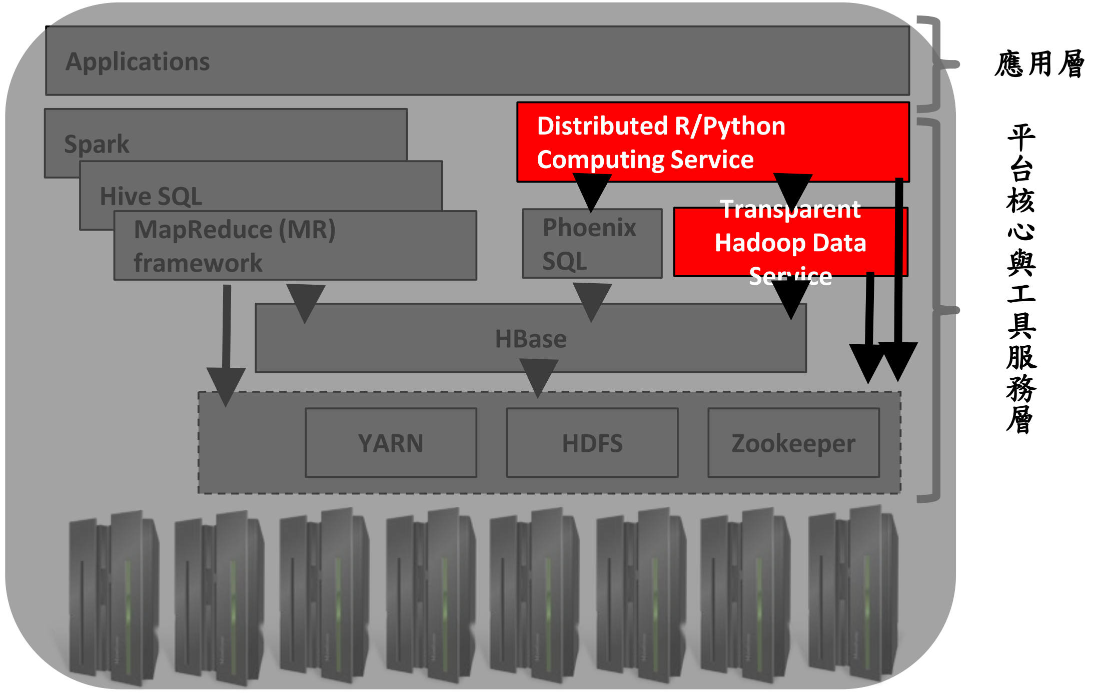
DRS 1.0及2.0執行之初任務失敗次數相當，然隨著時間的推進DRS 2.0學習任務所需記憶體需求量並調用能完成任務的計算單元，因而失敗次數逐漸下降

Interplay of DPS and TDS

DPS enjoys TDS: Per DPS container allocates data through TDS

DPS not only helps R/Python analytic programs to execute in a distributed, scalable platform, but allows the analytic programs to access data without sophisticated Hadoop knowledge

Both compute and storage are transparent to analytic users

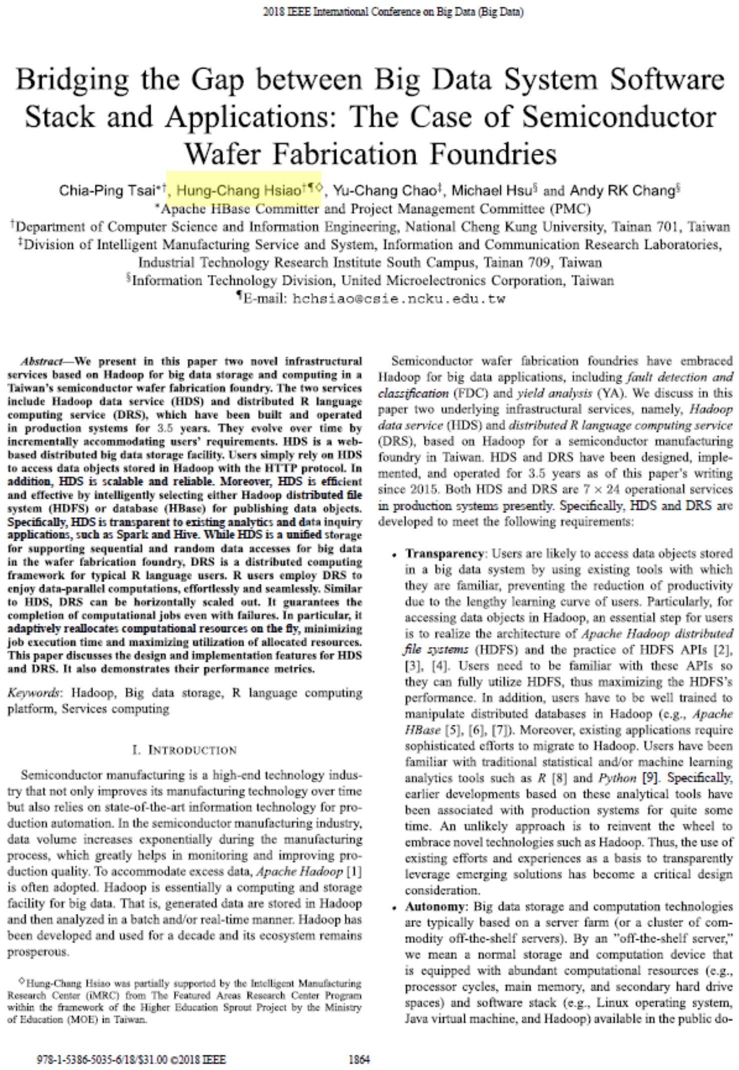


Summary and Outlook

- **TDS is used in a semiconductor manufacturing foundry for 3.5 yrs**
  - TDS服務: (k+1)-th storage: Hadoop (HDFS+HBase) + catalog
  - Features: Load balancing, synchronization based on sequential memory consistency model, intelligent and transparent data placement, compliance with HDFS interface and thus Hadoop ecosystem, pipelined and distributed data streaming, scalable and fault tolerant
  - 與HBase共生 for easy administration
- **DPS: Distributed R/Python computing service over TDS**
- **Contribute back to Apache Open Source community**
- **Ongoing: TDS and DPS in a low-end server box and/for streaming data**

Reference

- IEEE Big Data Conference 2018: IEEE 1st-tier conf. on big data, regular paper in the main track (100 accepted among 570 submissions, and 1100 participants), industry session



Bridging the Gap between Big Data System Software Stack and Applications: The Case of Semiconductor Wafer Fabrication Foundries

Chia-Ping Tsai[*][†], Hung-Chang Hsiao[†][¶][◇], Yu-Chang Chao[‡], Michael Hsu[§] and Andy RK Chang[§]
[*]Apache HBase Committer and Project Management Committee (PMC)
[†]Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan
[‡]Division of Intelligent Manufacturing Service and System, Information and Communication Research Laboratories, Industrial Technology Research Institute South Campus, Tainan 709, Taiwan
[§]Information Technology Division, United Microelectronics Corporation, Taiwan
[¶]E-mail: hchsiao@csie.ncku.edu.tw

***Abstract*—We present in this paper two novel infrastructural services based on Hadoop for big data storage and computing in a Taiwan's semiconductor wafer fabrication foundry. The two services include Hadoop data service (HDS) and distributed R language computing service (DRS), which have been built and operated in production systems for 3.5 years. They evolve over time by incrementally accommodating users' requirements. HDS is a web-based distributed big data storage facility. Users simply rely on HDS to access data objects stored in Hadoop with the HTTP protocol. In addition, HDS is scalable and reliable. Moreover, HDS is efficient and effective by intelligently selecting either Hadoop distributed file system (HDFS) or database (HBase) for publishing data objects. Specifically, HDS is transparent to existing analytics and data inquiry applications, such as Spark and Hive. While HDS is a unified storage for supporting sequential and random data accesses for big data in the wafer fabrication foundry, DRS is a distributed computing framework for typical R language users. R users employ DRS to enjoy data-parallel computations, effortlessly and seamlessly. Similar to HDS, DRS can be horizontally scaled out. It guarantees the completion of computational jobs even with failures. In particular, it adaptively reallocates computational resources on the fly, minimizing job execution time and maximizing utilization of allocated resources. This paper discusses the design and implementation features for HDS and DRS. It also demonstrates their performance metrics.**

Keywords: Hadoop, Big data storage, R language computing platform, Services computing

I. Introduction

Semiconductor manufacturing is a high-end technology industry that not only improves its manufacturing technology over time but also relies on state-of-the-art information technology for production automation. In the semiconductor manufacturing industry, data volume increases exponentially during the manufacturing process, which greatly helps in monitoring and improving production quality. To accommodate excess data, *Apache Hadoop* [1] is often adopted. Hadoop is essentially a computing and storage facility for big data. That is, generated data are stored in Hadoop and then analyzed in a batch and/or real-time manner. Hadoop has been developed and used for a decade and its ecosystem remains prosperous.

Semiconductor wafer fabrication foundries have embraced Hadoop for big data applications, including *fault detection and classification* (FDC) and *yield analysis* (YA). We discuss in this paper two underlying infrastructural services, namely, *Hadoop data service* (HDS) and *distributed R language computing service* (DRS), based on Hadoop for a semiconductor manufacturing foundry in Taiwan. HDS and DRS have been designed, implemented, and operated for 3.5 years as of this paper's writing since 2015. Both HDS and DRS are 7×24 operational services in production systems presently. Specifically, HDS and DRS are developed to meet the following requirements:

- **Transparency**: Users are likely to access data objects stored in a big data system by using existing tools with which they are familiar, preventing the reduction of productivity due to the lengthy learning curve of users. Particularly, for accessing data objects in Hadoop, an essential step for users is to realize the architecture of *Apache Hadoop distributed file systems* (HDFS) and the practice of HDFS APIs [2], [3], [4]. Users need to be familiar with these APIs so they can fully utilize HDFS, thus maximizing the HDFS's performance. In addition, users have to be well trained to manipulate distributed databases in Hadoop (e.g., *Apache HBase* [5], [6], [7]). Moreover, existing applications require sophisticated efforts to migrate to Hadoop. Users have been familiar with traditional statistical and/or machine learning analytics tools such as *R* [8] and *Python* [9]. Specifically, earlier developments based on these analytical tools have been associated with production systems for quite some time. An unlikely approach is to reinvent the wheel to embrace novel technologies such as Hadoop. Thus, the use of existing efforts and experiences as a basis to transparently leverage emerging solutions has become a critical design consideration.
- **Autonomy**: Big data storage and computation technologies are typically based on a server farm (or a cluster of commodity off-the-shelf servers). By an "off-the-shelf server," we mean a normal storage and computation device that is equipped with abundant computational resources (e.g., processor cycles, main memory, and secondary hard drive spaces) and software stack (e.g., Linux operating system, Java virtual machine, and Hadoop) available in the public do-

[◇]Hung-Chang Hsiao was partially supported by the Intelligent Manufacturing Research Center (iMRC) from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan.

SQL Loader

- **Objective: Provide SQL over Hadoop for analytic users**
- **SQL loader**
  - Load data stored in relational DBs into Hadoop (Phoenix/HBase)
- **Features:**
  - Multithreading (single machine) and MapReduce (cluster) versions
  - Incremental and fault tolerant

TDS- and DPS-Lite in a Server Box and/or for Streaming

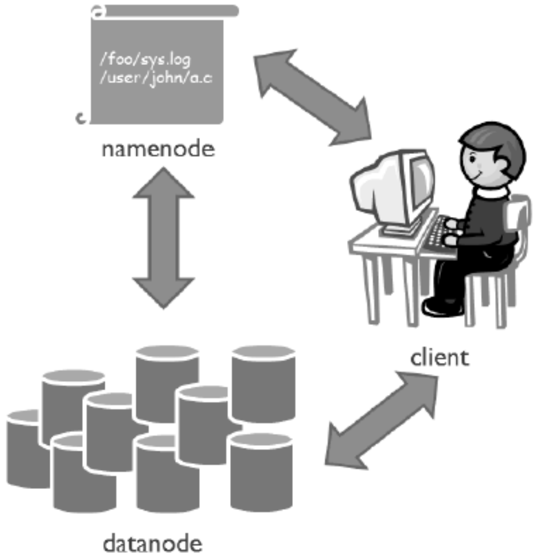
- **TDS and DPS server appliance**
  - Lite server (1 server) vs cluster farm (e.g., 30 servers)

- **Support for streaming**
  - Data continuously generated over time
  - TDS streams data to Hadoop
  - DPS computes streaming data

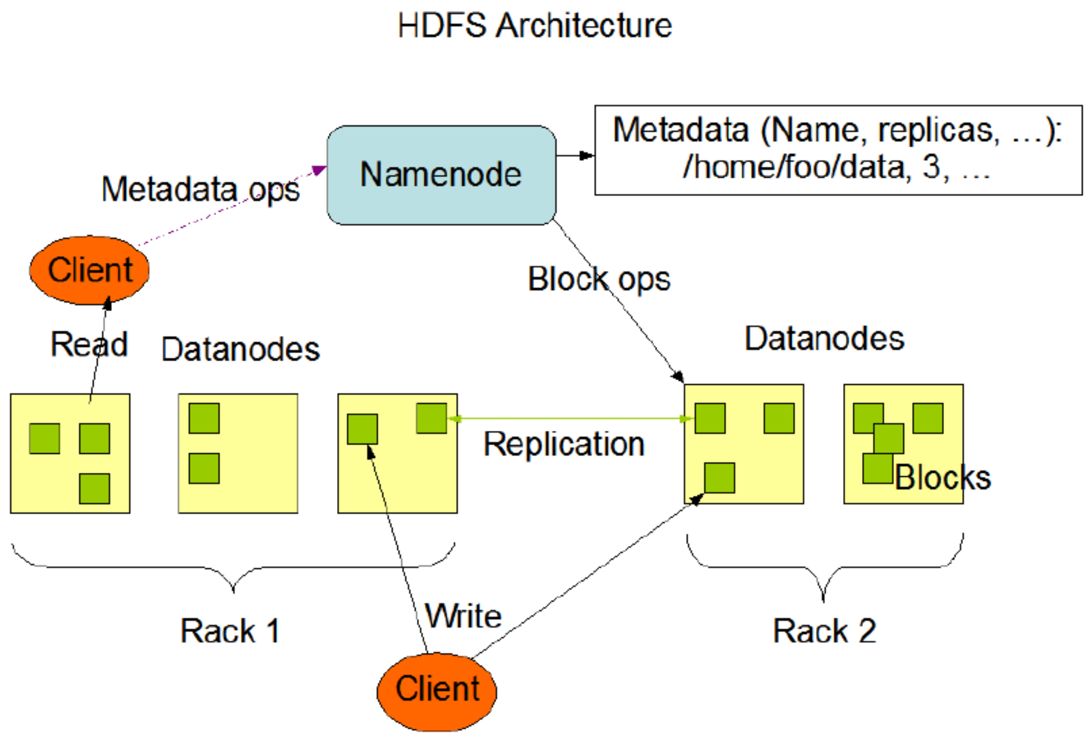
Thanks!

HDFS Architecture

- File is partitioned to fixed-size chunks
- Namenode manages a centralized directory for accesses like create, delete, append, etc.
    - Could have a backup standby
- Datanode stores file chunks
    - Datanodes may fail arbitrarily, and be added dynamically
    - Scale: $\times 10,000$
- Both namenode and datanode are capable of computation and storage (e.g., servers)



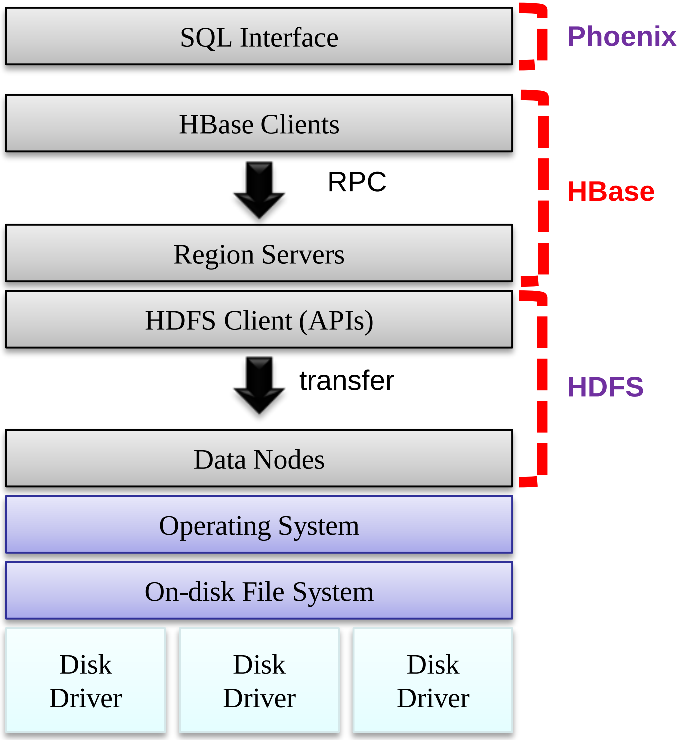
HDFS


HDFS Architecture
Metadata ops
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

What is HBase? (cont'd)

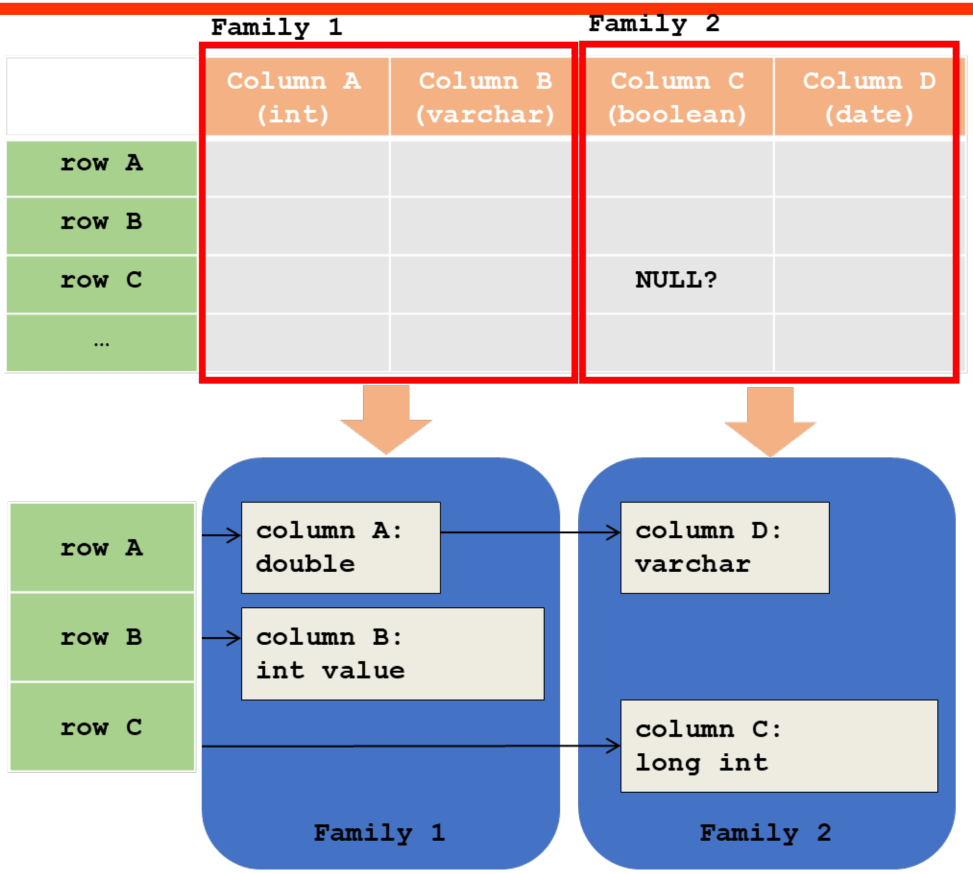
- Take advantages of commodity components (physical servers or VMs)

- Features:
  - Scalability
  - Availability
  - Reliability

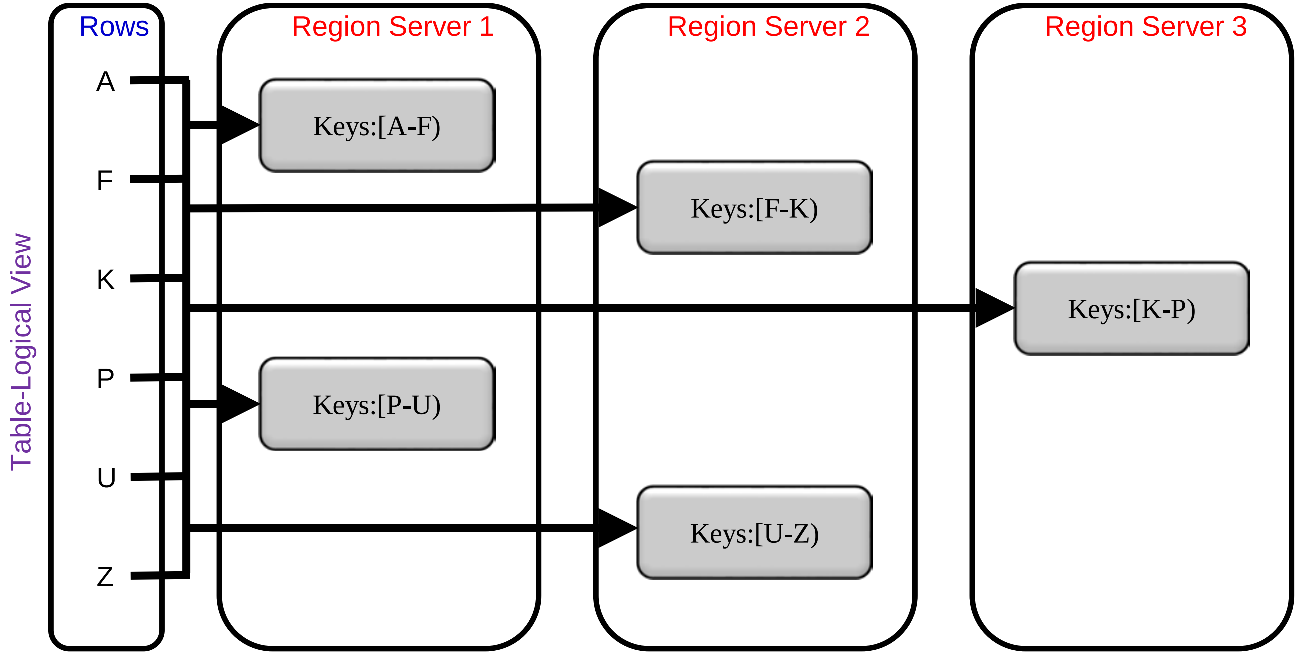


Component stacks

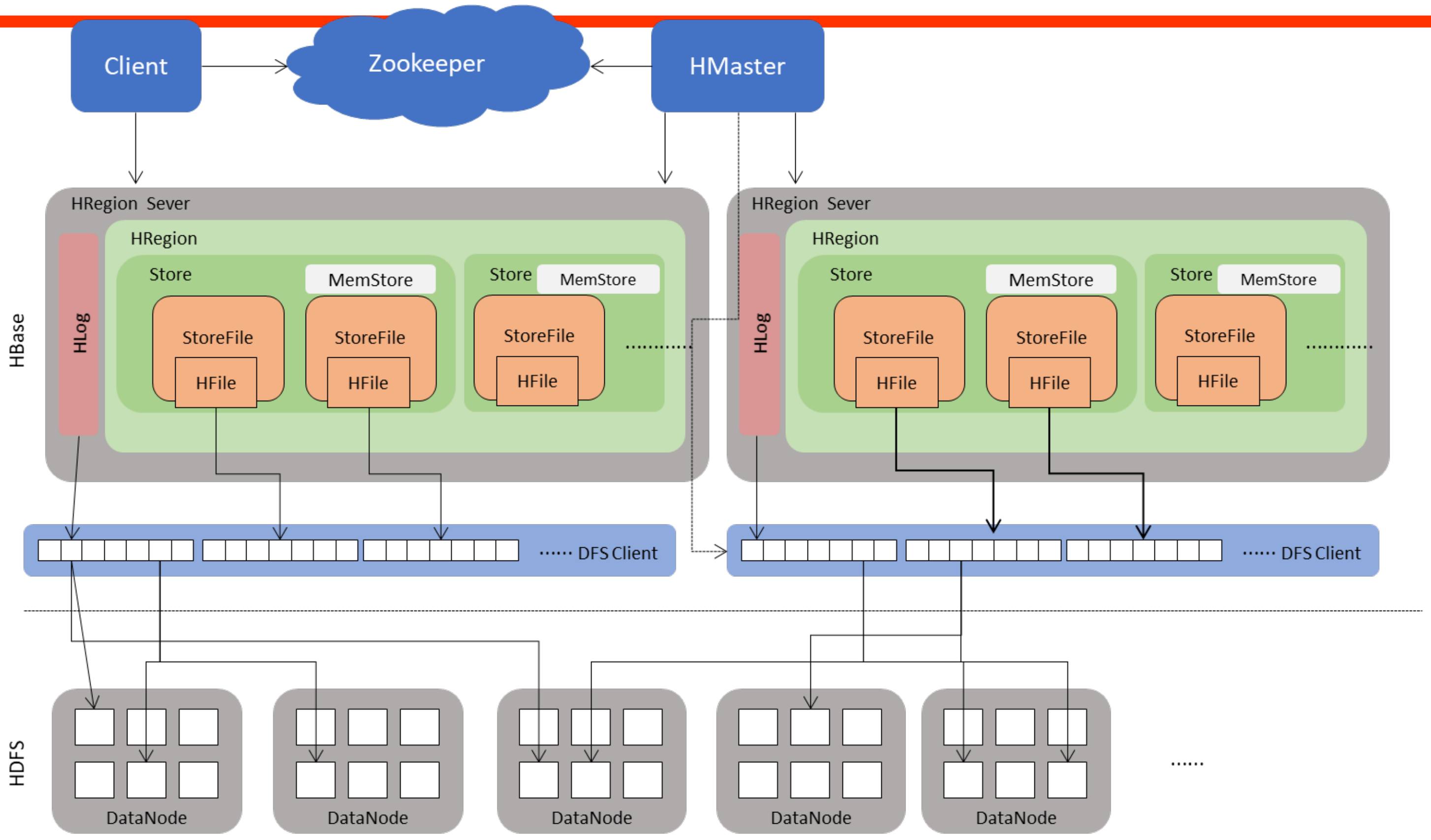
Overview (3/4): HBase Table



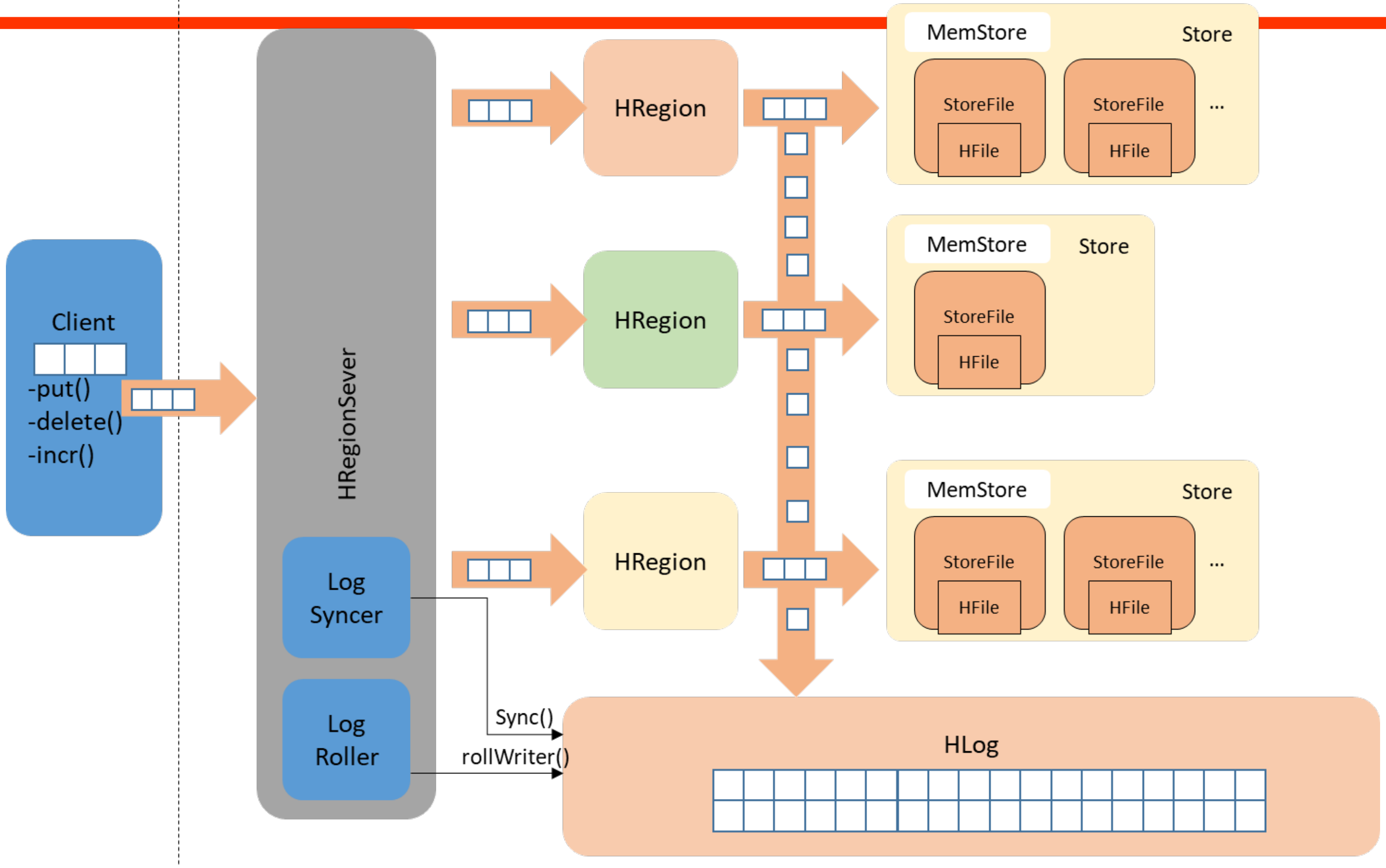
Architecture (2/7): Table and Region



Architecture (3/7): HRegionServer, HMaster and Zookeeper



Architecture (5/7): HLog, MemStore and HFile


Client
-put()
-delete()
-incr()
HRegionSever
Log Syncer
Log Roller
HRegion
HRegion
HRegion
MemStore
Store
StoreFile
HFile
StoreFile
HFile
...
MemStore
Store
StoreFile
HFile
MemStore
Store
StoreFile
HFile
StoreFile
HFile
...
Sync()
rollWriter()
HLog