Sector/Sphere Tutorial

Yunhong Gu

CloudCom 2010, Nov. 30, Indianapolis, IN

Outline

- Introduction to Sector/Sphere
- Major Features
- Installation and Configuration
- Use Cases

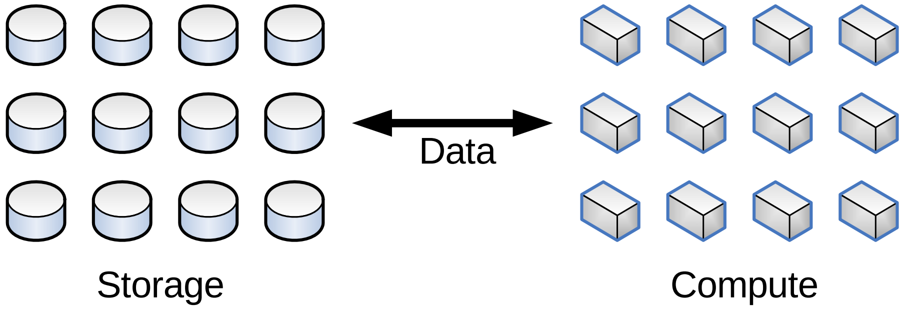
The Sector/Sphere Software

- Includes two components:
  - Sector distributed file system
  - Sphere parallel data processing framework

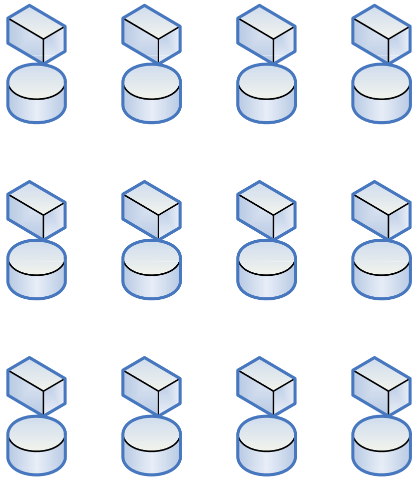
- Open Source, Developed in C++, Apache 2.0 license, available from http://sector.sf.net

- Started since 2006, current version is 2.5

Motivation: Data Locality

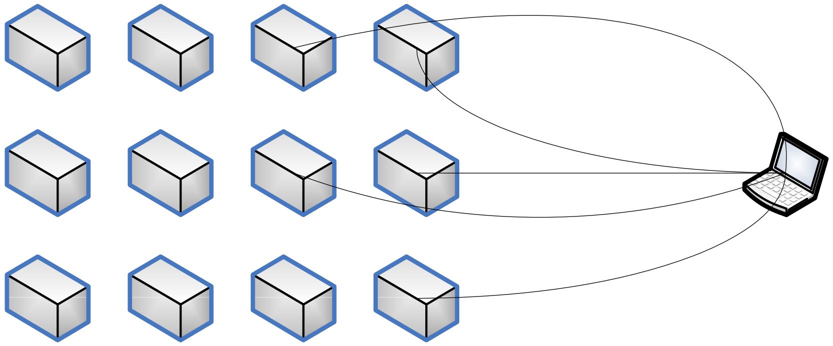


Traditional systems:
separated storage and computing sub-system
Expensive, data IO bandwidth bottleneck



Sector/Sphere model:
In-storage processing
Inexpensive, parallel data IO, data locality

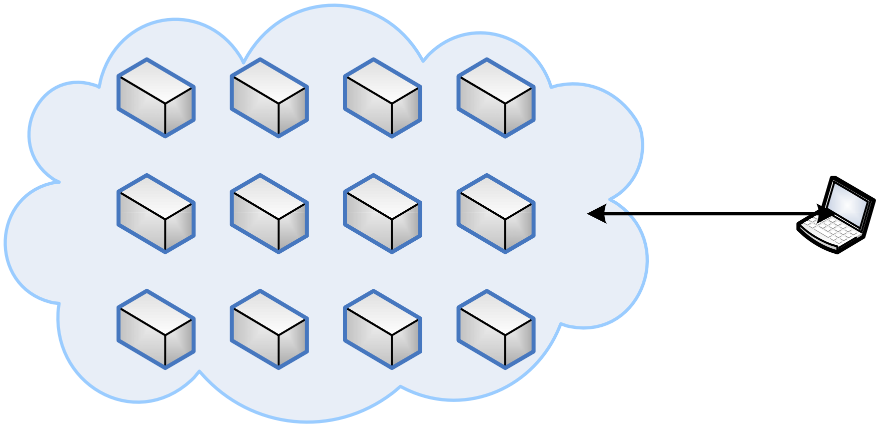
Motivation: Simplified Programming



Parallel/Distributed Programming with MPI, etc.:

Flexible and powerful.

very complicated application development

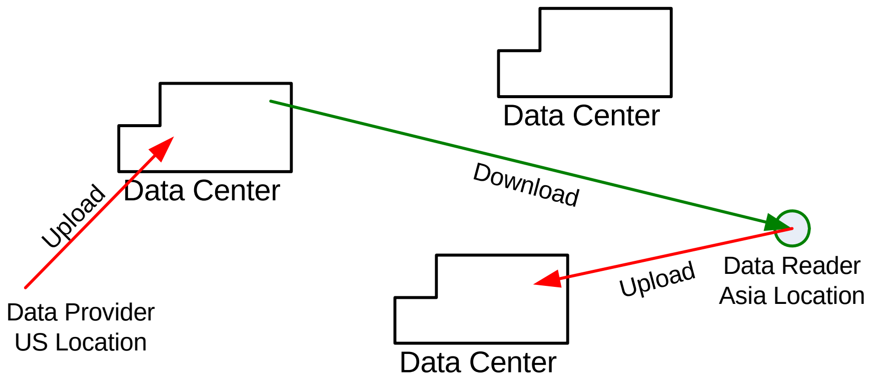


Sector/Sphere:

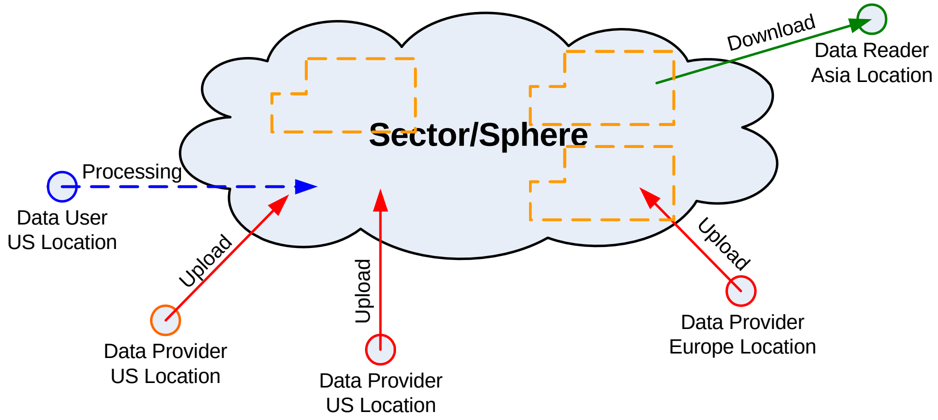
Clusters regarded as a single entity to the developer, simplified programming interface.

Limited to certain data parallel applications.

Motivation: Global-scale System

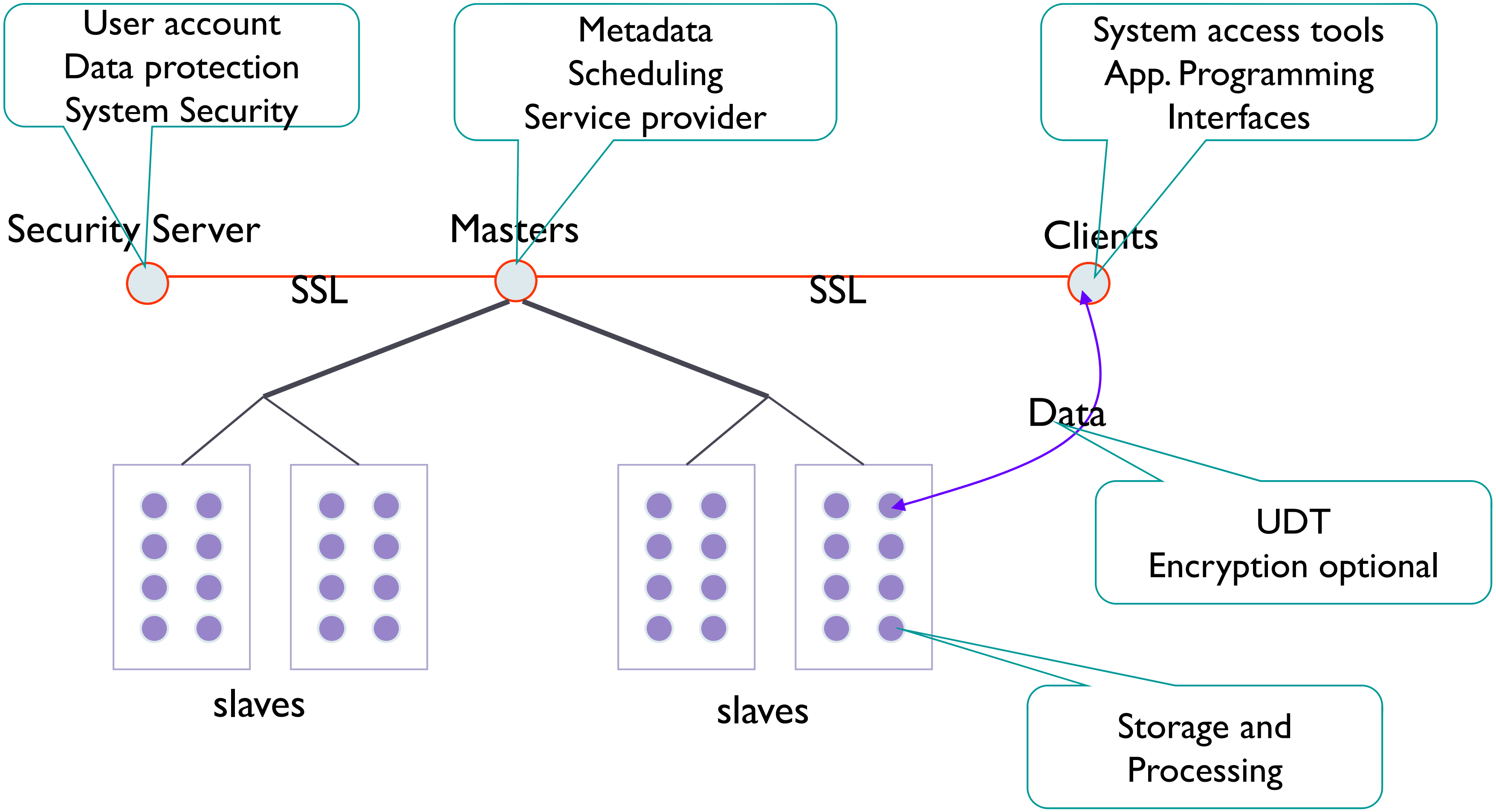


Traditional systems:
Require additional effort to locate and move data.



Sector/Sphere:
Support wide-area data collection and distribution.

Sector Distributed File System



Security Server

- User account authentication: password and IP address

- Sector uses its own account source, but can be extended to connected LDAP or local system accounts

- Authenticate masters and slaves with certificates and IP addresses

Master Server

- Maintain file system metadata

- Multiple active masters: high availability and load balancing
  - Can join and leave at run time
  - All respond to users' requests
  - Synchronize system metadata

- Maintain status of slave nodes and other master nodes

- Response users' requests

Slave Nodes

- Store Sector files
  - Sector is user space file system, each Sector file is stored on the local file system (e.g., EXT, XFS, etc.) of one or more slave nodes
  - Sector file is not split into blocks

- Process Sector data
  - Data is processed on the same storage node, or nearest storage node possible
  - Input and output are Sector files

Clients

- Sector file system client API
  - Access Sector files in applications using the C++ API
- Sector system tools
  - File system access tools
- FUSE
  - Mount Sector file system as a local directory
- Sphere programming API
  - Develop parallel data processing applications to process Sector data with a set of simple API

Topology Aware and Application Aware

- Sector considers network topology when managing files and scheduling jobs

- Users can specify file location when necessary, e.g., in order to improve application performance or comply with a security requirement.

Replication

- Sector uses replication to provide software level fault tolerance
  - No hardware RAID is required
- Replication number
  - All files are replicated to a specific number by default. No under-replication or over-replication is allowed.
  - Per file replication value can be specified
- Replication distance
  - By default, replication is created on furthest node
  - Per file distance can be specified, e.g., replication is created at local rack only.
- Restricted location
  - Files/directories can be limited to certain location (e.g., rack) only.

Fault Tolerance (Data)

- Sector guarantee data consistency between replicas
- Data is replicated to remote racks and data centers
  - Can survive loss of data center connectivity
- Existing nodes can continue to serve data no matter how many nodes are down
- Sector does not require permanent metadata; file system can be rebuilt from real data only

Fault Tolerance (System)

- All Sector master and slave nodes can join and leave at run time

- Master monitors slave nodes and can automatically restart a node if it is down; or remove a node if it appears to be problematic

- Clients automatically switch to good master/slave node if the current connected one is down
  - Transparent to users

UDT: UDP-based Data Transfer

- http://udt.sf.net
- Open source UDP based data transfer protocol
  - With reliability control and congestion control
- Fast, firewall friendly, easy to use
- Already used in many commercial and research systems for large data transfer
- Support firewall traversing via UDP hole punching

Wide Area Deployment

- Sector can be deployed across multiple data centers

- Sector uses UDT for data transfer

- Data is replicated to different data centers (configurable)
  - A client can choose a nearby replica
  - All data can survive even in the situation of losing connection to a data center

Rule-based Data Management

- Replication factor, replication distance, and restricted locations can be configured at per-file level and can be dynamically changed at run time

- Data IO can be balanced between throughput and fault tolerance at per client/per file level

In-Storage Data Processing

- Every storage node is also a compute node
- Data is processed at local node or the nearest available node
- Certain file operations such as md5sum and grep can run significantly faster in Sector
  - In-storage processing + parallel processing
  - No data IO is required
- Large data analytics with Sphere and MapReduce API

Summary of Sector’s Unique Features

- Scale up to 1,000s of nodes and petabytes of storage
- Software level fault tolerance (no hardware RAID is required)
- Works both within a single data center or across distributed data centers with topology awareness
- In-storage massive parallel data processing via Sphere and MapReduce APIs
- Flexible rule-based data management
- Integrated WAN acceleration
- Integrated security and firewall traversing features
- Integrated system monitoring

Limitations

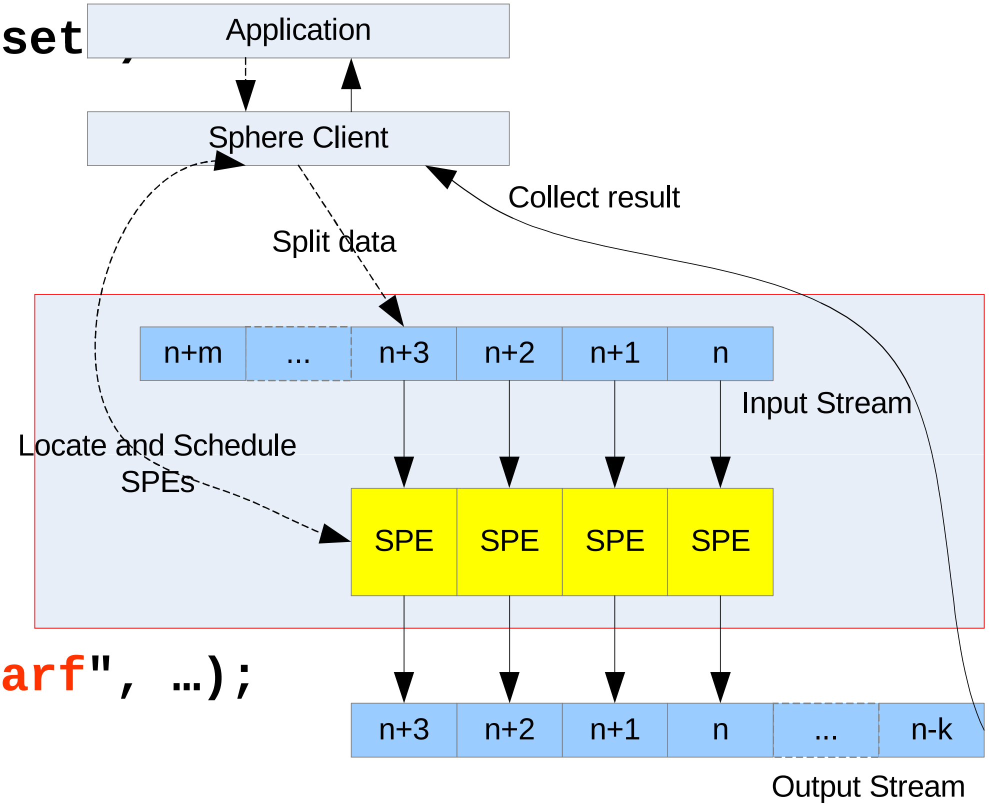
- File size is limited by available space of individual storage nodes.
- Users may need to split their datasets into proper sizes.
- Sector is designed to provide high throughput on large datasets, rather than extreme low latency on small files.

Sphere: Simplified Data Processing

- Data parallel applications
- Data is processed at where it resides, or on the nearest possible node (locality)
- Same user defined functions (UDF) are applied on all elements (records, blocks, files, or directories)
- Processing output can be written to Sector files or sent back to the client
- Transparent load balancing and fault tolerance

Sphere: Simplified Data Processing

```
for each file F in (SDSS dataset
  for each image I in F
     findBrownDwarf(I, …);
```

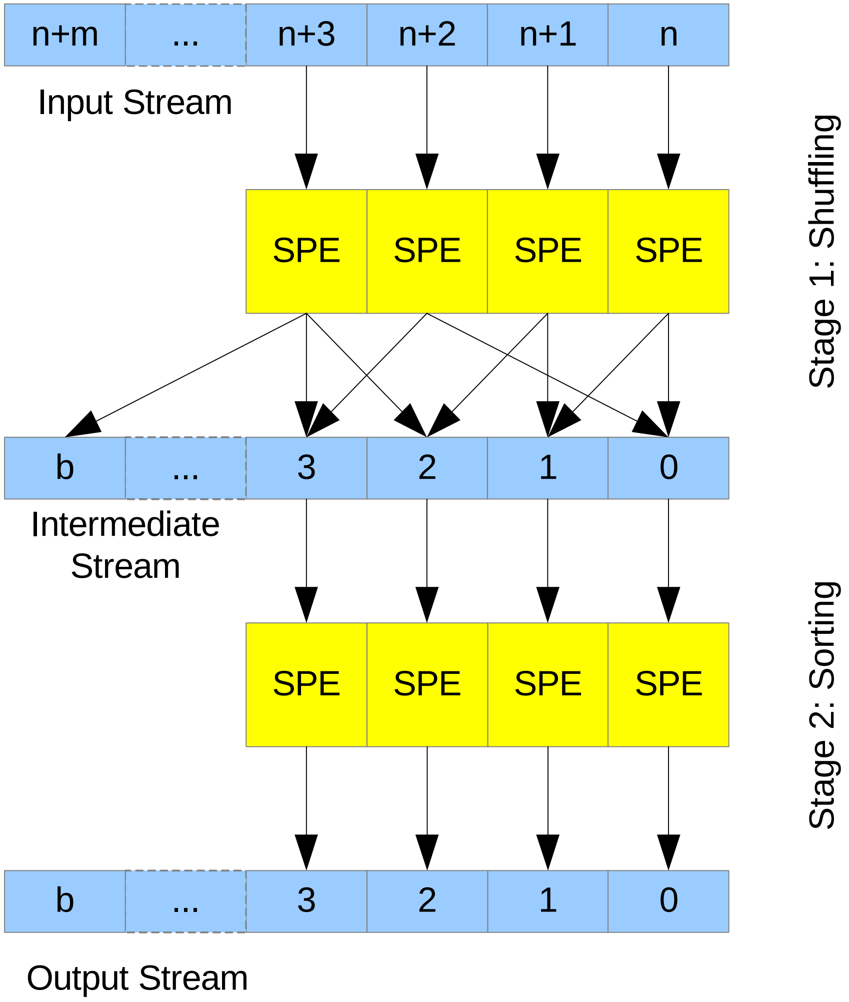


```
SphereStream sdss;
sdss.init("sdss files");
SphereProcess myproc;
myproc->run(sdss,"findBrownDwarf", …);
```

```
findBrownDwarf(char* image, int isize, char* result, int rsize);
```

Sphere: Data Movement

- Slave -> Slave Local
- Slave -> Slaves (Hash/Buckets)
  - Each output record is assigned an ID; all records with the same ID are sent to the same "bucket" file
- Slave -> Client



What does a Sphere program like?

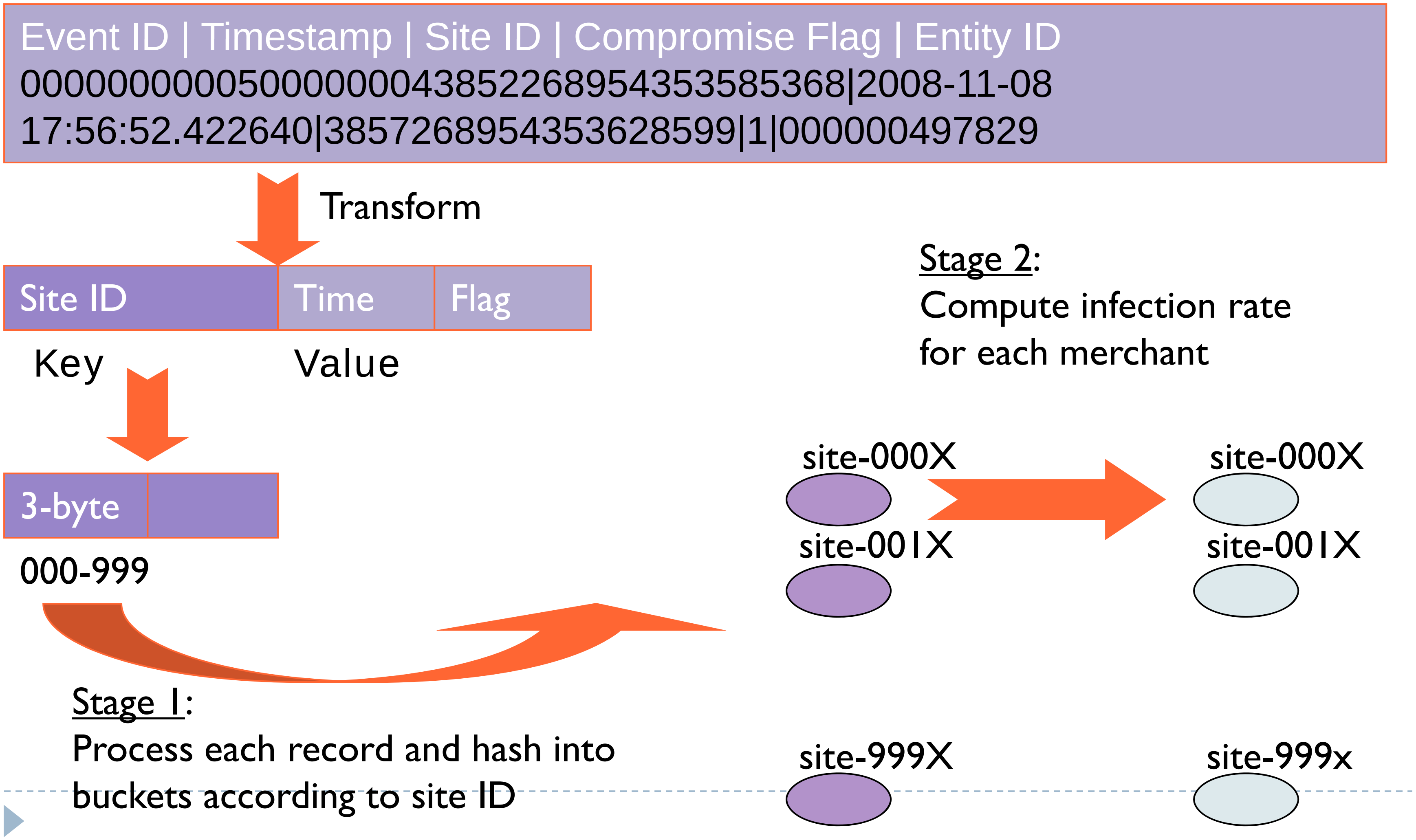
- A client application
  - Specify input, output, and name of UDF
  - Inputs and outputs are usually Sector directories or collection of files
  - May have multiple round of computation if necessary (iterative/combinative processing)
- One or more UDFs
  - C++ functions following the Sphere specification (parameters and return value)
  - Compiled into a dynamic library (*.so)

The MalStone Benchmark

- Drive-by problem: visit a web site and get comprised by malware.
- MalStone-A: compute the infection ratio of each site.
- MalStone-B: compute the infection ratio of each site from the beginning to the end of every week.

http://code.google.com/p/malgen/

MalStone



Text Record

Event ID | Timestamp | Site ID | Compromise Flag | Entity ID
0000000000050000000043852268954353585368|2008-11-08
17:56:52.422640|3857268954353628599|1|000000497829

Transform

Stage 1:
Process each record and hash into
buckets according to site ID

Stage 2:
Compute infection rate
for each merchant

MalStone code

- Input: collection of log files
- UDF-1
  - Read a log file, process each line, obtain the site ID, and hash it into a bucket ID, generate a new record by filtering out unnecessary information
- Intermediate result: bucket files, each file containing information on a subset of sites
- UDF-2:
  - Read a bucket file, compute the infection ratio, per site and per week
- Output: Files containing infection ratios per site

Prepare for Installation

- Download:
  - http://sourceforge.net/projects/sector
- Documentation:
  - http://sector.sourceforge.net/doc/index.htm
- Linux, g++ 4.x, openssl-dev, fuse (optional)
  - Windows porting in progress
- In a testing system, all components can run on the same machine

Code Structure

- conf : configuration files
- doc: Sector documentation
- examples: Sphere programming examples
- fuse: FUSE interface
- include: programming header files (C++)
- lib: places to stored compiled libraries
- master: master server
- tools: client tools
- security: security server
- slave: slave server
- Makefile

Compile/Make

- Download sector.2.5.tar.gz from Sector SourceForge project website
- tar –zxvf sector.2.5.tar.gz
- cd ./sector-sphere; make
- RPM package is also available

Configuration

- **./conf/master.conf**: master server configurations, such as Sector port, security server address, and master server data location
- **./conf/slave.conf**: slave node configurations, such as master server address and local data storage path
- **./conf/client.conf**: master server address and user account/password so that a user doesn't need to specify this information every time they run a Sector tool

Configuration File Path

- $SECTOR_HOME/conf
- ../conf
  - If $SECTOR_HOME is not set, all commands should be run at their original directory
- /opt/sector/conf (RPM installation)

- #SECTOR server port number
- #note that both TCP/UDP port N and N-1 will be used
- **SECTOR_PORT**
- **6000**

- #security server address
- **SECURITY_SERVER**
- **ncdm153.lac.uic.edu:5000**

- #data directory, for the master to store temporary system data
- #this is different from the slave data directory and will not be used to store data files
- **DATA_DIRECTORY**
- **/home/u2/yunhong/work/sector_master/**

- #number of replicas of each file, default is 1
- **REPLICA_NUM**
- **2**

Start and Stop Server (Testing)

- Run all sector servers on the same node

- Start Security Server
  - ./security/sserver

- Start Master server
  - ./master/start_master

- Start Slave server
  - ./slave/start_slave

Start and Stop Sector (Real)

- Step 1: start the security server ./security/sserver.
  - Default port is 5000, use *sserver new_port* for a different port number

- Step 2: start the masters and slaves using ./master/start_all
  - #1. distribute master certificate to all slaves
  - #2. configure password-free ssh from master to all slave nodes
  - #3. configure ./conf/slaves.list

- To shutdown Sector, use ./master/stop_all (brutal force) or ./tools/sector_shutdown (graceful)

Check the Installation

- At ./tools, run sector_sysinfo
- This command should print the basic information about the system, including masters, slaves, files in the system, available disk space, etc.
- If nothing is displayed or incorrect information is displayed, something is wrong.
- It may be helpful to run “start_master” and “start_slave” manually (instead of “start_all”) in order to debug

Sector Client Tools

- Located at ./tools
- Most file system commands are available: ls, stat, rm, mkdir, mv, etc.
  - Note that Sector is a user space file system and there is no mount point for these commands. Absolute dir has to be passed to the commands.
- Wild cards * and ? are supported

Upload/Download

- sector_upload can be used to load files into Sector
- sector_upload <src file/dir> <dst dir> [-n num_of_replicas] [-a ip_address] [-c cluster_id] [--e(ncryption)]

- sector_download can be used to download data to local file system
- sector_download <sector_file/dir> <local_dir> [--e]

- You can run these over Internet connections, benefiting from the integrated UDT WAN acceleration

Sector-FUSE

- Require FUSE library installed

- ./fuse
  - make
  - ./sector-fuse <local path>

- FUSE allows Sector to be mounted as a local file system directory so you can use the common file system commands to access Sector files.

SectorFS API

- C++ API

- You may open any source files in ./tools as an example for SectorFS API.

- Sector requires login/logout, init/close.

- File operations are similar to common FS APIs, e.g., open, read, write, seekp/seekg, tellp/tellg, close, stat, etc.

Sphere API

- C++ API for both Sphere UDF and MapReduce interface
- Learn By Example: see example applications in sector-sphere/examples.
  - Most examples are within 100 – 200 lines of C++ code
- Documentation of each API is also available
  - http://sector.sourceforge.net/doc/index.htm

Use Scenario #1

- Use Sector as distributed data storage/manage system

- Sector is inexpensive (open source, commodity hardware), very scalable, support high availability with multiple active masters, high performance IO with direct data access

- Few other file systems can
  - Support wide area deployments with single instance
  - Support dynamic per-file data management rules

- Reasonable security

Use Scenario #2

- Sector can be used as an advanced data sharing platform

- It can aggregate large number of geographically distributed servers with a unified namespace
- Nearby replica can be chosen for more bandwidth
- UDT enables high speed data transfer from remote clients

- Compare to FTP or other point-to-point/one-to-many systems
  - Single data server vs. 1000s of data servers
  - TCP/HTTP vs. UDT
  - Single point of failure vs. fault tolerance
  - Centralized servers vs. distributed servers

Use Scenario #3

- Sector/Sphere can be used for high performance large data analytics

- Comparable to Hadoop MapReduce
- Faster than Hadoop by 2 – 4x

For More Information

- Project Website: http://sector.sf.net
- SourceForge: http://sourceforge.net/projects/sector
- Contact me: Yunhong Gu first_name.last_name@gmail